- Closed form of KL for Gaussians:
$$D_{KL} = \frac{1}{2}\Big[\log\frac{\det(\boldsymbol{\Sigma}_2)}{\det(\boldsymbol{\Sigma}_1)} - d + \operatorname{tr}(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1) + (\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)^\top\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)\Big].$$
- Cosine theorem:
$$\|\boldsymbol{x}-\boldsymbol{y}\|^2 = \|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 - 2\langle\boldsymbol{x},\boldsymbol{y}\rangle.$$
- Points of $\mathcal{S}$ are in general position if any subset $\Xi \subseteq \mathcal{S}$ with $|\Xi| \le \delta$ is linearly independent.

Connectionism

McCulloch-Pitts neuron

$$f[\boldsymbol{\sigma},\theta](\boldsymbol{x}) \doteq \mathbb{1}\{\boldsymbol{\sigma}^\top\boldsymbol{x} \ge \theta\}, \quad \boldsymbol{\sigma}\in\{-1,1\}^n, \boldsymbol{x}\in\{0,1\}^n, \theta\in\mathbb{R}.$$

Perceptron

$$f[\boldsymbol{w},b](\boldsymbol{x}) \doteq \operatorname{sgn}(\boldsymbol{w}^\top\boldsymbol{x}+b)$$
$$\gamma[\boldsymbol{w},b](\boldsymbol{x},y) \doteq \frac{y(\boldsymbol{w}^\top\boldsymbol{x}+b)}{\|\boldsymbol{w}\|}$$
$$\gamma[\boldsymbol{w},b](\mathcal{S}) \doteq \min_{(\boldsymbol{x},y)\in\mathcal{S}} \gamma[\boldsymbol{w},b](\boldsymbol{x},y)$$
$$\mathcal{V}(\mathcal{S}) \doteq \{(\boldsymbol{w},b) \mid \gamma[\boldsymbol{w},b](\mathcal{S}) > 0\}.$$

Decision boundary hyperplane: $\boldsymbol{w}^\top\boldsymbol{x}/\|\boldsymbol{w}\| + b/\|\boldsymbol{w}\| \stackrel{!}{=} 0$. If $\gamma[\boldsymbol{w},b](\mathcal{S}) > 0$, then $\mathcal{S}$ is linearly separated. $\mathcal{V}(\mathcal{S}) \neq \emptyset$ iff $\mathcal{S}$ is linearly separable. Adding points to $\mathcal{S}$ makes $\mathcal{V}(\mathcal{S})$ smaller.

The perceptron algorithm tries to find any $(\boldsymbol{w},b) \in \mathcal{V}(\mathcal{S})$. It does not aim to find solution with smaller error if $\mathcal{V}(\mathcal{S}) = \emptyset$. Iterative mistake-driven algorithm:
$$f[\boldsymbol{w},b](\boldsymbol{x}) \neq y \implies \boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}, \quad b \leftarrow b + y.$$
For all iterates $\boldsymbol{w}_t \in \operatorname{span}(\boldsymbol{x}_1,\dots,\boldsymbol{x}_n)$.

Convergence can be proven by using $\boldsymbol{w}^\top\boldsymbol{w}_t \ge t\gamma$ and $\|\boldsymbol{w}_t\| \le R\sqrt{t}$, where $\|\boldsymbol{w}\| = 1$, $\gamma[\boldsymbol{w}](\mathcal{S}) > 0$, and $R = \max_{\boldsymbol{x}\in\mathcal{S}}\|\boldsymbol{x}\|$. Then, bound $1 \ge \cos\angle(\boldsymbol{w},\boldsymbol{w}_t) = \boldsymbol{w}^\top\boldsymbol{w}_t/\|\boldsymbol{w}\|\|\boldsymbol{w}_t\|$. We convergence within $\lfloor R^2/\gamma^2 \rfloor$ iterations.

Number of unique classifications:
$$\mathcal{C}(\mathcal{S},d) = \left|\left\{\boldsymbol{y}\in\{-1,+1\}^n \;\middle|\; \exists\boldsymbol{w} : \forall i : y_i(\boldsymbol{w}^\top\boldsymbol{x}_i) > 0\right\}\right|$$
Assume that points are in general position. Cover's theorem:
$$\mathcal{C}(n+1,d) = 2\sum_{i=0}^{d-1}\binom{n}{i}.$$
Proof: Base cases are both 2 and adding a point has two cases → Recurrence:
$$\mathcal{C}(n+1,d) = \mathcal{C}(n,d) + \mathcal{C}(n,d-1).$$
For $n \le d$, we have $\mathcal{C}(n,d) = 2^n$. After $n = 2d$, there is a steep decrease in number of linear classification, quickly moving toward 0.

Parallel distributed processing

(1) A set of processing units with states of activation; (2) Output functions for each unit; (3) A pattern of connectivity between units; (4) Propagation rules for propagating patterns of activity; (5) Activation functions for units; (6) A learning rule to modify connectivity based on experience; (7) An environment within which the system must operate.

Hopfield networks

Models an associative memory, which aims to reconstruct a memory from an input that has been subjected to noise. Energy function via second-order interactions between n binary neurons:
$$H(\boldsymbol{x}) \doteq \frac{1}{2}\sum_{i=1}^d\sum_{j=1}^d w_{ij}x_ix_j + \sum_{i=1}^n b_ix_i, \quad \boldsymbol{x}\in\{-1,1\}^d.$$
We have $w_{ii} = 0$ and $w_{ij} = w_{ji}$. Simple dynamics:
$$x_i \leftarrow \begin{cases}+1 & H([.,x_{i-1},+1,x_{i+1},.]) \le H([.,x_{i-1},-1,x_{i+1},.])\\ -1 & \text{otherwise.}\end{cases}$$
Or: $x_i \leftarrow \operatorname{sgn}(H_i)$, where $H_i \doteq \sum_{j=1}^d w_{ij}x_j - b_i$.

Hebbian learning (neurons frequently in the same state reinforce):
$$\boldsymbol{W} = \frac{1}{d}\sum_{t=1}^n \boldsymbol{x}_t\boldsymbol{x}_t^\top.$$
Pattern $\boldsymbol{x}_t$ is memorized if meta-stable: update rule makes no updates to it:
$$x_{ti} = \operatorname{sgn}(x_{ti} + C_{ti}), \quad C_{ti} \doteq \frac{1}{d}\sum_{j=1}^d\sum_{r\neq t}^n x_{ri}x_{rj}x_{tj}.$$
If cross-talk $|C_{ti}| < 1$ for all i, then $\boldsymbol{x}_t$ is meta-stable.

Feedforward networks

Regression models

Mean-squared error
$$\ell[\boldsymbol{\theta}](\mathcal{S}) = \frac{1}{2}\sum_{i=1}^n(f[\boldsymbol{\theta}](\boldsymbol{x}_i) - y_i)^2.$$
Linear model: $\ell[\boldsymbol{w}](\mathcal{S}) = \frac{1}{2}\|\boldsymbol{X}^\top\boldsymbol{w} - \boldsymbol{y}\|^2 \to$ Closed form solution: $\boldsymbol{w}^\star = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$.

Logistic regression (binary outputs) use sigmoid: $\sigma(z) \doteq 1/1+\exp(-z)$. Binary cross-entropy loss:
$$\ell[\boldsymbol{\theta}](\mathcal{S}) = -\frac{1}{n}\sum_{i=1}^n y_i\log\sigma(f[\boldsymbol{\theta}](\boldsymbol{x}_i)) + (1-y_i)\log(1-f[\boldsymbol{\theta}](\boldsymbol{x}_i)).$$
(Multi-class) Cross-entropy loss:
$$\ell[\boldsymbol{\theta}](\boldsymbol{x},y) = -\log\operatorname{softmax}(f[\boldsymbol{\theta}](\boldsymbol{x}))_y.$$

Layers and units

Mapping: $f[\boldsymbol{W},\boldsymbol{b}](\boldsymbol{x}) = \phi(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b})$, where ϕ is a pointwise activation function. DNNs compose: $G = F_L[\boldsymbol{\theta}_L]\circ\cdots\circ F_1[\boldsymbol{\theta}_1]$.

Intermediate layers are permutation symmetric: $\boldsymbol{F}[\boldsymbol{W},\boldsymbol{b}](\boldsymbol{x}) = \boldsymbol{P}^{-1}\phi(\boldsymbol{P}\boldsymbol{W}\boldsymbol{x}+\boldsymbol{P}\boldsymbol{b}) = \boldsymbol{P}^{-1}F[\boldsymbol{P}\boldsymbol{W},\boldsymbol{P}\boldsymbol{b}](\boldsymbol{x}) \to$ Parameters are not unique.

Linear networks

Linear layers are closed under composition → We do not gain representational power from composing them.

Residual networks

Residual layers:
$$F[\boldsymbol{W},\boldsymbol{b}](\boldsymbol{x}) = \boldsymbol{x} + \phi(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) - \phi(\boldsymbol{0}).$$
Then, $F[\boldsymbol{0},\boldsymbol{0}] = \mathrm{Id}$. This makes it such that the model learns how to incrementally learn a better representation, rather than having to learn it at every layer. If input and output have different dimensionality, linearly project $\boldsymbol{x}$. Using this architecture, the depth can be increased significantly, because gradients propagate better.

Sigmoid networks

MLP with sigmoid activation:
$$g[\boldsymbol{v},\boldsymbol{W},\boldsymbol{b}](\boldsymbol{x}) \doteq \boldsymbol{v}^\top\sigma(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}), \quad \boldsymbol{v},\boldsymbol{b}\in\mathbb{R}^m, \boldsymbol{W}\in\mathbb{R}^{m\times n}.$$
Function class: $\mathcal{G}_n \doteq \bigcup_{m=1}^\infty \mathcal{G}_{n,m}$, where
$$\mathcal{G}_{n,m} \doteq \{g[\boldsymbol{v},\boldsymbol{W},\boldsymbol{b}] \mid \boldsymbol{b}\in\mathbb{R}^m, \boldsymbol{W}\in\mathbb{R}^{m\times n}\}.$$
Or, as a linear span of units,
$$\mathcal{G}_n = \operatorname{span}\{\sigma(\boldsymbol{w}^\top\boldsymbol{x}+b) \mid \boldsymbol{w}\in\mathbb{R}^n, b\in\mathbb{R}\}.$$
This set universally approximates $\mathcal{C}(\mathbb{R}^n)$. But, it does not provide insight into how depth affects performance → Barron: Let $f : \mathbb{R}^n \to \mathbb{R}$ with finite $\mathcal{C}_f$ and any $r > 0$, there is a sequence of one hidden layer MLPs $(g_m)_{m\in\mathbb{N}}$ such that
$$\int_{r\mathbb{B}}(f(\boldsymbol{x}) - g_m(\boldsymbol{x}))^2\mu(\mathrm{d}\boldsymbol{x}) \le \mathcal{O}(1/m),$$
where $r\mathbb{B}$ is a r-radius ball and μ is any probability measure. Relaxing the notion of approximation to squared error over ball with radius r gives a decay of $1/m$ on the approximation error.

ReLU networks

ReLU: $(z)_+ \doteq \max_{0,z}$. Consider a layer of m ReLU units, then each unit is active or inactive: $\mathbb{1}\{\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b} > 0\} \in \{0,1\}^m$. As such, we can partition the input space into cells that have the same activation pattern:
$$\mathcal{X}_{\boldsymbol{\kappa}} \doteq \{\boldsymbol{x} \mid \mathbb{1}\{\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b} > 0\} = \boldsymbol{\kappa}\}.$$
The number of cells is a proxy for the complexity of a network. Consider a ReLU network with L layers of $m > n$ width. The number of linear regions is lower bounded by
$$R(m,L) \ge R(m)\lfloor m/n\rfloor^{n(L-1)}.$$
Thus, complexity is related to depth.

Piecewise linear functions are approximators of $\mathcal{C}(\mathbb{R})$ and a piecewise linear function g with m pieces can be written as
$$g(x) = ax + b + \sum_{i=1}^{m-1} c_i(x - x_i)_+.$$
Using the lifting lemma, ReLU networks are universal approximators of $\mathcal{C}(\mathbb{R})$.

Gradient-based learning

Backpropagation

Backpropagation computes gradient in linear time if we know the gradient of all basic blocks in the function. Assume we have the following function,
$$F[\boldsymbol{\theta}](\boldsymbol{x}) \doteq (F_L\circ\cdots\circ F_1)(\boldsymbol{x}), \quad \boldsymbol{h}_\ell \doteq F_\ell[\boldsymbol{\theta}_\ell](\boldsymbol{h}_{\ell-1}), \quad \boldsymbol{h}_0 = \boldsymbol{x}.$$
We need the following,
$$\boldsymbol{\delta}_\ell = \frac{\partial h(\boldsymbol{\theta})}{\partial \boldsymbol{h}_\ell}.$$
We have a recurrence,
$$\boldsymbol{\delta}_\ell = \left[\frac{\partial \boldsymbol{h}_{\ell+1}}{\partial \boldsymbol{h}_\ell}\right]^\top\boldsymbol{\delta}_{\ell+1}, \quad \boldsymbol{\delta}_L = \frac{\partial h(\boldsymbol{\theta})}{\partial \boldsymbol{h}_L}.$$
Then, to compute parameter gradients,
$$\frac{\partial h(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_\ell} = \boldsymbol{\delta}_\ell\frac{\partial \boldsymbol{h}_\ell}{\partial \boldsymbol{\theta}_\ell}.$$

Gradient descent

Update rule:
$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta\nabla_{\boldsymbol{\theta}}h(\boldsymbol{\theta}^t), \quad \eta > 0, \quad h(\boldsymbol{\theta}) \doteq \ell\circ F[\boldsymbol{\theta}].$$
Discretization of ODE:
$$\mathrm{d}\boldsymbol{\theta} = -\nabla h(\boldsymbol{\theta})\mathrm{d}t.$$
Trajectory outcome (point where $\nabla h(\boldsymbol{\theta}) = \boldsymbol{0}$) depends on initial conditions.

Gradient descent can only be successful if gradients change slowly → Smoothness: h is L-smooth if
$$\|\nabla h(\boldsymbol{\theta}) - \nabla h(\boldsymbol{\theta}')\| \le L\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \quad \forall\boldsymbol{\theta},\boldsymbol{\theta}'.$$
Or: $\|\nabla^2 h(\boldsymbol{\theta})\|_2 \le L$ for all $\boldsymbol{\theta}$. If $\eta = 1/L$, then we have sufficient decrease,
$$h(\boldsymbol{\theta}') \le h(\boldsymbol{\theta}) - \frac{1}{2L}\|\nabla h(\boldsymbol{\theta})\|^2, \quad \forall\boldsymbol{\theta},\boldsymbol{\theta}'.$$
Let h be L-smooth and $\eta = 1/L$, then an ϵ-critical point will be found in at most
$$T = \frac{2L}{\epsilon^2}(h(\boldsymbol{\theta}^0) - h(\boldsymbol{\theta}^\star)).$$
Proof: Sufficient decrease → Telescoping sum and min $\le \sum$.

h satisfies PL-inequality with $\mu > 0$ if
$$\frac{1}{2}\|\nabla h(\boldsymbol{\theta})\|^2 \ge \mu(h(\boldsymbol{\theta}) - h(\boldsymbol{\theta}^\star)), \quad \forall\boldsymbol{\theta}.$$
Intuition: If $\boldsymbol{\theta}$ has small gradient, then it is near-optimal. Let h be L-smooth and μ-PL and $\eta = 1/L$, then
$$h(\boldsymbol{\theta}_T) - h(\boldsymbol{\theta}^\star) \le \left(1 - \frac{\mu}{L}\right)^T(h(\boldsymbol{\theta}_0) - h(\boldsymbol{\theta}^\star)).$$
Proof: Sufficient decrease → PL → Subtract $h(\boldsymbol{\theta}^\star)$ both sides.

Acceleration, adaptivity, and momentum

Nesterov acceleration achieves better theoretical guarantees than GD:
$$\boldsymbol{\chi}^{t+1} = \boldsymbol{\theta}^t + \gamma(\boldsymbol{\theta}^t - \boldsymbol{\theta}^{t-1}), \quad \boldsymbol{\theta}^{t+1} = \boldsymbol{\chi}^{t+1} - \eta\nabla h(\boldsymbol{\chi}^{t+1}).$$
Momentum intuition: If gradient is stable, we can make bolder steps. Heavy Ball:
$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta\nabla h(\boldsymbol{\theta}^t) + \beta(\boldsymbol{\theta}^t - \boldsymbol{\theta}^{t-1}), \quad \beta\in[0,1].$$
Assuming constant gradient $\boldsymbol{\delta}$, we have
$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta\left(\sum_{\tau=1}^{t-1}\beta^\tau\right)\boldsymbol{\delta}.$$
Thus, learning rate increases in case of a stable gradient.

Adaptivity intuition: Different parameters behave differently → Parameter-specific learning rates:
$$\theta_i^{t+1} = \theta_i^t - \eta_i^t\partial_i h(\boldsymbol{\theta}^t)$$
$$\eta_i^t \doteq \frac{\eta}{\sqrt{\gamma_i^t + \delta}}, \quad \gamma_i^t \doteq \gamma_i^{t-1} + [\partial_i h(\boldsymbol{\theta}^t)]^2.$$
Parameters with small gradient magnitude will have a larger step size.

Adam combines adaptivity and momentum:
$$\boldsymbol{g}_t = \beta\boldsymbol{g}_{t-1} + (1-\beta)\nabla h(\boldsymbol{\theta}_t), \quad \beta\in[0,1]$$
$$\boldsymbol{\gamma}_t = \alpha\boldsymbol{\gamma}_{t-1} + (1-\alpha)\nabla h(\boldsymbol{\theta}_t)^{\odot 2}, \quad \alpha\in[0,1]$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{\eta}_t\odot\boldsymbol{g}_t, \quad \boldsymbol{\eta}_t = 1\oslash(\sqrt{\boldsymbol{\gamma}_t} + \delta).$$
$\boldsymbol{g}_t$ is a smooth gradient estimator and $\boldsymbol{\gamma}_t$ measures the stability of the optimization landscape.

Stochastic gradient descent

When the dataset is too large, computing the full gradient is infeasible → Estimate gradient with a mini-batch (SGD). SGD outperforms GD in practice, because it has a lower chance of getting stuck in a local optimum due to variance in the gradient estimator.

Convolutional networks

Convolution

Integral operator:
$$(Tf)(u) \doteq \int_{t_1}^{t_2} H(u,t)f(t)\mathrm{d}t.$$
Fourier transform:
$$(\mathcal{F}f)(u) \doteq \int_{-\infty}^{\infty} e^{-2\pi itu}f(t)\mathrm{d}t.$$
Convolution:
$$(f*h)(u) \doteq \int_{-\infty}^{\infty} h(u-t)f(t)\mathrm{d}t.$$
Commutative: $f*h = h*f$, Shift-equivariant: $f_\Delta * h = (f*h)_\Delta$, Convolution as Fourier: $f*h = \mathcal{F}^{-1}(\mathcal{F}f\cdot\mathcal{F}h)$. All proofs are done by defining new variables dependent on existing ones. Linear shift-equivariant operator $\iff$ Convolution.

Discrete convolution:
$$(f*h)[u] = \sum_{t=-\infty}^{\infty} f[t]h[u-t].$$
Cross-correlation:
$$(f\star h)[u] = \sum_{t=-\infty}^{\infty} f[t]h[u+t].$$
Toeplitz matrix $\boldsymbol{H}_n^h \in \mathbb{R}^{(n+m-1)\times n}$
$$\boldsymbol{H}_n^h \doteq \begin{bmatrix} h_1 & 0 & \cdots & 0 & 0\\ h_2 & h_1 & \cdots & 0 & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & \cdots & h_m & h_{m-1}\\ 0 & 0 & \cdots & 0 & h_m\end{bmatrix}.$$
This can then be applied to a vectorized $\boldsymbol{f}\in\mathbb{R}^n$. Effectively a proof that convolution is linear with increased statistical efficiency.

Convolutional networks

Images are 2D, so use 2D definition of convolution:
$$(\boldsymbol{X}*\boldsymbol{W})[i,j] = \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} x_{i-k,j-l}w_{k,l}.$$
Let $\boldsymbol{X}^\ell$ be output of the ℓ-th convolutional layer and $\boldsymbol{\Delta}^\ell \doteq \frac{\partial h}{\partial \boldsymbol{X}^\ell}$, then
$$\boldsymbol{\Delta}^{\ell-1} = \boldsymbol{\Delta}^\ell\star\boldsymbol{W}^\ell, \quad \frac{\partial h}{\partial \boldsymbol{W}^\ell} = \boldsymbol{\Delta}^\ell * \boldsymbol{X}^{\ell-1}.$$
Max-pooling layer:
$$x_{ij}^\ell = \max\{x_{i+k,j+l}^{\ell-1} \mid k,l\in[0,r)\}$$
$$\frac{\partial x_{ij}^\ell}{\partial x_{mn}^{\ell-1}} = \mathbb{1}\{(m,n) = (i^\star,j^\star)\},$$
where $(i^\star,j^\star)$ are the indices of maximum value in the forward pass.

Data generally has multiple channels:
$$(\mathbf{X}*\mathbf{W})[c,i,j] = \sum_{r=1}^{C_{in}}\sum_{k=-K}^{K}\sum_{l=-K}^{K} w_{c,r,k,l}x_{r,i-k,j-l}.$$
Fully connected in channels and local in spatial dimensions.

Recurrent neural networks

Activations are computed recursively to handle variable-length data,

$$\boldsymbol{z}_t = F[\boldsymbol{\theta}](\boldsymbol{z}_{t-1}, \boldsymbol{x}_t).$$

Dependent on application, compute output variables:

$$\boldsymbol{y}_t = G[\boldsymbol{\varphi}](\boldsymbol{z}_t).$$

Elman RNN:

$$F[\boldsymbol{U},\boldsymbol{V}](\boldsymbol{z},\boldsymbol{x}) = \phi(\boldsymbol{U}\boldsymbol{z}+\boldsymbol{V}\boldsymbol{x}), \quad G[\boldsymbol{W}](\boldsymbol{z}) = \psi(\boldsymbol{W}\boldsymbol{z}).$$

Bidirectional RNNs do both ways and concatenate. Stacked RNNS connect layers horizontally:

$$\boldsymbol{z}_{t,l} = \phi(\boldsymbol{U}_l \boldsymbol{z}_{t-1,l} + \boldsymbol{V}_l \boldsymbol{z}_{t,l-1}), \quad \boldsymbol{z}_{t,0} = \boldsymbol{x}_t.$$

Let $L = \sum_{t=1}^{T} \ell(\hat{\boldsymbol{y}}_t, \boldsymbol{y}_t)$, then we have gradients:

$$\frac{\partial L}{\partial \boldsymbol{U}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{U}}, \quad \frac{\partial L}{\partial \boldsymbol{V}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{V}},$$

where

$$\frac{\partial L}{\partial \boldsymbol{z}_t} = \sum_{i=t}^{T} \frac{\partial \ell(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i)}{\partial \hat{\boldsymbol{y}}_i} \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{z}_i} \prod_{j=t+1}^{i} \dot{\boldsymbol{\Phi}}_j \boldsymbol{U},$$

where $\dot{\Phi}_j = \operatorname{diag}(\phi'(\boldsymbol{U}\boldsymbol{z}_{j-1} + \boldsymbol{V}\boldsymbol{x}_j))$. This is only stable if $\|\dot{\boldsymbol{\Phi}}_j \boldsymbol{U}\|_2 = 1$, which is almost never the case → exploding/vanishing gradient.

Gated memory

Solve vanishing gradient by gating.

LSTM ($\boldsymbol{z}_t$: cell state, $\boldsymbol{\zeta}_t$: hidden state):

$$\boldsymbol{z}_t = \sigma(\boldsymbol{F}\tilde{\boldsymbol{x}}_t) \odot \boldsymbol{z}_{t-1} + \sigma(\boldsymbol{G}\tilde{\boldsymbol{x}}_t) \odot \tanh(\boldsymbol{V}\tilde{\boldsymbol{x}}_t)$$

$$\tilde{\boldsymbol{x}}_t = [\boldsymbol{\zeta}_{t-1}, \boldsymbol{x}_t]$$

$$\boldsymbol{\zeta}_t = \sigma(\boldsymbol{H}\tilde{\boldsymbol{x}}_t) \odot \tanh(\boldsymbol{U}\boldsymbol{z}_t).$$

GRU simplifies the LSTM to only 3 weight matrices:

$$\boldsymbol{z}_t = \boldsymbol{\sigma} \odot \boldsymbol{z}_{t-1} + (1-\boldsymbol{\sigma}) \odot \tilde{\boldsymbol{z}}_t, \quad \boldsymbol{\sigma} = \sigma(\boldsymbol{G}\tilde{\boldsymbol{x}}_t)$$

$$\tilde{\boldsymbol{x}}_t = [\boldsymbol{z}_{t-1}, \boldsymbol{x}_t]$$

$$\tilde{\boldsymbol{z}}_t = \tanh(\boldsymbol{V}[\boldsymbol{\zeta}_t \odot \boldsymbol{z}_{t-1}, \boldsymbol{x}_t]), \quad \boldsymbol{\zeta}_t = \sigma(\boldsymbol{H}[\boldsymbol{z}_{t-1}\boldsymbol{x}_t]).$$

Linear recurrent model

Simplify GRU to be linear such that it can exploit prefix scan parallelism which has $\mathcal{O}(\log T)$ runtime:

$$\boldsymbol{z}_t = \boldsymbol{\sigma} \odot \boldsymbol{z}_{t-1} + (1-\boldsymbol{\sigma}) \odot \tilde{\boldsymbol{z}}_t, \quad \boldsymbol{\sigma} = \sigma(\boldsymbol{G}\boldsymbol{x}_t), \quad \tilde{\boldsymbol{z}}_t = \boldsymbol{V}\boldsymbol{x}_t.$$

We can ensure that the gradients do not explode by parametrizing the GRU smartly. The LRU has hidden state evolution in a discrete time linear system:

$$\boldsymbol{z}_{t+1} = \boldsymbol{A}\boldsymbol{z}_t + \boldsymbol{B}\boldsymbol{x}_t.$$

Diagonalize $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$, where $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_m), \lambda_i \in \mathbb{C}$. This linear system is stable if the modulus of the eigenvalues is bounded by 1. Thus, we parameterize them such that the moduli can only be in $(0,1)$ in the following way,

$$\lambda_i = \exp(-\exp(\nu_i))(\cos(\phi_i) + \sin(\phi_i)i).$$

This uses $\exp(\theta i) = \cos(\theta) + \sin(\theta)i$ and that we can represent complex numbers in polar coordinate form via modulus r and phase ϕ:

$$z = r(\cos(\phi)\sin(\phi)i), \quad r = |z|.$$

Thus, $r_i = \exp(-\exp(\nu_i)) \in (0,1)$. Thus, at initialization we sample

$$\phi_i \sim \text{Unif}([0, 2\pi]), \quad r_i \sim \text{Unif}(I), \quad I \subseteq [0,1].$$

And compute $\nu_i = \log(-\log r_i)$. The modulus always remains upper bounded by 1.

We do not lose any representational power, because we can put all representational power into the output map:

$$\boldsymbol{y}_t = \text{MLP}(\text{Re}(\boldsymbol{G}\boldsymbol{\zeta}_t)), \quad \boldsymbol{G} \in \mathbb{C}^{k \times m}.$$

Sequence learning

Generate sequence step-by-step, given another sequence:

$$p(\boldsymbol{y}_{1:n} \mid \boldsymbol{x}_{1:m}) = \prod_{i=1}^{n} p(y_i \mid \boldsymbol{y}_{1:i-1}, \boldsymbol{x}_{1:m}).$$

This is done by mapping input sequence to latent representation (encoder RNN):

$$x_1, \dots, x_m \mapsto \boldsymbol{\zeta}.$$

Decoder RNN uses this along with history to predict next token:

$$\boldsymbol{\zeta}, y_1, \dots, y_{t-1} \mapsto \boldsymbol{z}_{t-1}.$$

This is mapped to a distribution over next tokens,

$$\boldsymbol{z}_{t-1} \mapsto \boldsymbol{\mu}_t, \quad y_t \sim p(\boldsymbol{\mu}_t).$$

Problem: Lossy compression of input sequence. Solution: Use attention such that decoder can look at full input sequence at every step:

$$a_{ij} = \text{softmax}_j(\text{MLP}([\boldsymbol{z}_{i-1}, \boldsymbol{\zeta}_j])), \quad \boldsymbol{c}_t = \sum_{i=1}^{m} a_{ti} \boldsymbol{\zeta}_i.$$

This allows for alignment between input and output sequence.

Transformers

Self-attention

Let $\boldsymbol{X} \in \mathbb{R}^{T \times d}$ denote a sequence of input embeddings. Problem: They are non-contextual. Self-attention:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V,$$

where $\boldsymbol{W}_Q, \boldsymbol{W}_K \in \mathbb{R}^{d \times d_k}$ and $\boldsymbol{W}_V \in \mathbb{R}^{d \times d_v}$. Attention:

$$\boldsymbol{\Xi} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}.$$

Multi-headed self-attention performs attention h times in parallel.

Cross-attention

Two sequences $\boldsymbol{A} \in \mathbb{R}^{T_a \times d_a}$, $\boldsymbol{B} \in \mathbb{R}^{T_b \times d_b}$ and we want to give information of $\boldsymbol{B}$ to $\boldsymbol{A}$:

$$\boldsymbol{Q} = \boldsymbol{A}\boldsymbol{W}_Q, \quad \boldsymbol{K} = \boldsymbol{B}\boldsymbol{W}_K, \quad \boldsymbol{V} = \boldsymbol{B}\boldsymbol{W}_V,$$

where $\boldsymbol{W}_Q \in \mathbb{R}^{d_a \times d_k}$, $\boldsymbol{W}_K \in \mathbb{R}^{d_b \times d_k}$, and $\boldsymbol{W}_V \in \mathbb{R}^{d_b \times d_v}$.

Positional encoding

Attention is permutation equivariant → Add positional encoding matrix $\boldsymbol{P} \in \mathbb{R}^{T \times d}$ with

$$p_{tk} = \begin{cases} \sin(t\omega_k) & k \mod 2 = 0 \\ \cos(t\omega_k) & k \mod 1 = 0, \end{cases} \quad \omega_k \doteq C^{k/d}.$$

Machine translation

Encoder-decoder architecture where the encoder applies MHSA to input sequence and decoder applies masked MHSA to history and then cross-attention with contextualized input sequence. Furthermore, MLP, Layer normalization, and residual layers are used.

BERT

BERT is a transformer-based pretrained language model that is used for finetuning on downstream NLP tasks. BERT tokenizes uses WordPiece tokenization and prepends a `[CLS]` token. When the weights of the encoders are pretrained, we can place additional layers on top that operate on the contextualized BERT tokens.

Two pre-training stages:

1. Predicting masked out tokens using its left and right context as input (Cloze test; trains BERT's understanding of language);
2. Binary next sentence classification, where the model must classify two sentences as being consecutive or not (trains BERT to infer relationships between sentences).

Vision transformer

ViT adapts the transformer to images by treating projected 16×16 image patches as tokens. A possible reason for this model's effectiveness is that this architecture carries less inductive bias than CNN-based models. In general, this seems to be beneficial for very large datasets.

Geometric deep learning

GDL models neural networks that satisfy invariances by design.

Invariance and equivariance

f (arbitrary number of inputs) is order-invariant iff

$$f(\boldsymbol{X}) = f(\boldsymbol{P}\boldsymbol{X}), \quad \boldsymbol{X} \in \mathbb{R}^{M \times d},$$

where $\boldsymbol{P}$ is a permutation matrix.

f (arbitrary number of inputs and same number of outputs) is equivariant iff

$$f(\boldsymbol{X}) = \boldsymbol{P}f(\boldsymbol{P}\boldsymbol{X}).$$

We want models that have these properties by design.

Deep sets

Let $\phi : R \to \mathbb{R}^d$ be a pointwise feature extractor network. Deep Sets obtains an order-invariant representation of the input set by summing their features up. This representation can be given to any network $\rho : \mathbb{R}^d \to \mathcal{Y}$:

$$f(\boldsymbol{x}_1, \dots, \boldsymbol{x}_M) = \rho\left(\sum_{m=1}^{M} \phi(\boldsymbol{x}_m)\right).$$

We can easily turn this into an equivariant map by providing $\boldsymbol{x}_m$ to $\rho : R \times \mathbb{R}^d \to \mathcal{Y}$:

$$f(\boldsymbol{x}_1, \dots, \boldsymbol{x}_M)_i = \rho\left(\boldsymbol{x}_i, \sum_{m=1}^{M} \phi(\boldsymbol{x}_m)\right).$$

This architecture is universal for a fixed d, but it requires mapping that are highly discontinuous for $M \to \infty$.

PointNet

PointNet is a Deep Sets architecture on three-dimensional point clouds. The model employs T-net blocks, which apply rigid transformations to the point cloud, which is permutation invariant. These are applied twice alternatingly with MLPs to form ϕ. This gives a 64-dim intermediate feature vector and 1024-dim final feature vector. The features are aggregated by a max-pool operator.

Object classification: ρ is an MLP with a softmax head that takes the global feature vector as input.

Object segmentation: ρ concatenates intermediate local feature and final global feature, which is given to MLP with softmax head.

Graph neural networks

Let $\boldsymbol{A}$ be the adjacency matrix of an undirected graph. f is order-invariant on a graph if

$$f(\boldsymbol{X}, \boldsymbol{A}) = f(\boldsymbol{P}\boldsymbol{X}, \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^\top).$$

And equivariant if

$$f(\boldsymbol{X}, \boldsymbol{A}) = \boldsymbol{P}f(\boldsymbol{P}\boldsymbol{X}, \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^\top).$$

Let $\boldsymbol{X}_m = \{\{\boldsymbol{x}_n \mid a_{nm} = 1\}\}$ (multiset of neighbors' features). ϕ takes $\boldsymbol{x}_m$ and $\boldsymbol{X}_m$ as input (any pair of isomorphic graphs result in same feature representations):

$$\phi(\boldsymbol{x}_m, \boldsymbol{X}_m) = \phi\left(\boldsymbol{x}_m, \bigoplus_{\boldsymbol{x} \in \boldsymbol{X}_m} \psi(\boldsymbol{x})\right).$$

This is a message-passing scheme.

Graph convolutional networks (GCNs) aggregates local neighborhoods with a fixed set of weights (coupling matrix),

$$\bar{\boldsymbol{A}} \doteq \boldsymbol{D}^{-1/2}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}^{-1/2}, \quad \boldsymbol{D} = \operatorname{diag}(\boldsymbol{d}), \quad d_m = 1 + \sum_{n=1}^{M} a_{nm}.$$

Now, $\bar{\boldsymbol{A}}\boldsymbol{X}$ computes the average feature over neighbors and the node itself. GCNs introduce learnable parameters $\boldsymbol{W}$:

$$f[\boldsymbol{W}](\boldsymbol{X}, \boldsymbol{A}) = \sigma(\bar{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}).$$

This is an equivariant function, which can be stacked. Then, we can use the final representations to do graph classification or node classification.

Limitations: Requires a depth equal to diameter of graph to exchange information between all nodes; In very deep GCNs, node features become indistinguishable due to smoothing of $\bar{\boldsymbol{A}}$; Bottleneck effect of how much information can be stored in fixed-size representations. There are no canonical solutions to these problems.

GATs introduce attention (which is equivariant) in the neighborhood function and replaces $\bar{\boldsymbol{A}}$. It does so by parametrizing the coupling matrix $\boldsymbol{Q}$,

$$q_{mn} = \text{softmax}_n(\rho(\boldsymbol{u}^\top[\boldsymbol{V}\boldsymbol{x}_m, \boldsymbol{V}\boldsymbol{x}_n, \boldsymbol{x}_{mn}]))$$

$$\sum_{n=1}^{M} a_{mn} q_{mn} = 1, \quad \forall m \in [M].$$

Here, $\boldsymbol{x}_{ij}$ is an edge feature.

Despite better adaptivity, GATs are still message-passing algorithms. Such algorithms have inherent limitations in the type of graphs they can distinguish. The Weisfeiler-Lehman graph isomorphism test computes whether there exists an isomorphism between two graphs. It can be shown that GCNs and GATs cannot distinguish graphs beyond the WL-test.

Spectral graph theory

Laplacian operator measures local deviation from the mean in vanishingly small neighborhoods:

$$\Delta f = \sum_{i=1}^{d} \frac{\partial^2 f}{\partial x_i^2}.$$

Graph Laplacian: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$. Degree-normalized Laplacian: $\tilde{\boldsymbol{L}} = \boldsymbol{D}^{-1/2}(\boldsymbol{D} - \boldsymbol{A})\boldsymbol{D}^{-1/2}$. We can generalize Fourier transform to graphs: Diagonalize $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$ and $\boldsymbol{U}$ can be seen as graph Fourier basis and $\boldsymbol{\Lambda}$ as frequencies. Graph convolution can be computed as pointwise multiplication in Fourier domain:

$$\boldsymbol{X} * \boldsymbol{Y} = \boldsymbol{U}((\boldsymbol{U}^\top \boldsymbol{X}) \odot (\boldsymbol{U}^\top \boldsymbol{Y})).$$

This can be learned by

$$G[\boldsymbol{\theta}](\boldsymbol{L})\boldsymbol{X} = \boldsymbol{U}G[\boldsymbol{\theta}](\boldsymbol{\Lambda})\boldsymbol{U}^\top \boldsymbol{X}.$$

Problem: Eigendecomposition of $\boldsymbol{L}$ takes $\mathcal{O}(M^3)$. Solution: Use polynomial kernels:

$$\boldsymbol{U}\left(\sum_{k=0}^{K} \alpha_k \boldsymbol{\Lambda}\right)\boldsymbol{U}^\top \boldsymbol{X} = \sum_{k=1}^{K} \alpha_k \boldsymbol{L}^k \boldsymbol{X}.$$

Here, the polynomial order K defines the kernel size (or neighborhood size). $\boldsymbol{\alpha} \in \mathbb{R}^K$ are parameters.

Tricks of the trade

Initialization

Parameters are typically chosen with a fixed variance by sampling from

$$\boldsymbol{\theta} \sim \mathcal{N}(0, \sigma^2), \quad \boldsymbol{\theta} \sim \text{Unif}([-\sqrt{3}\sigma, \sqrt{3}\sigma]).$$

LeCun init, $\sigma = 1/\sqrt{n}$, preserves input variance. Glorot init, $\sigma = \sqrt{2/n+m}$, normalizes magnitude of gradient (intuition: backpropagation combines n-dim input and m-dim output). Kaiming init, $\sigma = \sqrt{2/n}$, is designed to be used with ReLU by observing that only half of the units are active in expectation. Orthogonal init considers the weights holistically by initializing as orthogonal matrix (benefit: eigenvalues are ± 1).

Weight decay

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta(\boldsymbol{\nabla} h(\boldsymbol{\theta}_t) - \mu\boldsymbol{\theta}_t) = (1-\eta\mu)\boldsymbol{\theta}_t - \boldsymbol{\nabla} h(\boldsymbol{\theta}_t).$$

Equivalent to (1) gradient descent with ℓ_2 regularization, (2) Bayesian prior that weights are sampled from normal distribution, (3) Lagrangian with constraint $\|\boldsymbol{\theta}\| \leq \mu$.

Under the basis of the Hessian's eigenvectors, the optimum of ℓ_2-regularized ℓ_μ is

$$\boldsymbol{\theta}_\mu^\star = \operatorname{diag}\left(\frac{\lambda_i}{\lambda_i + \mu}\right)\boldsymbol{\theta}^\star.$$

Each axis is scaled based on sensitivity. If $\lambda_i \gg \mu$, then $\lambda_i/\lambda_i+\mu \approx 1$, so the solution does not change much.

Early stopping

Stop if validation performance does not improve for past p epochs. Crudely equivalent to weight decay with μ if stopped at $T \approx 1/\eta\mu$.

Dropout

Randomly disable subset of parameters during training → Units become less dependent on one another → Instead of units being specialized, they become generally useful.

Two view: (1) regularization and (2) ensemble of networks:

$$p[\boldsymbol{\theta}](\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{\boldsymbol{b} \in \{0,1\}^P} p(\boldsymbol{b}) p[\boldsymbol{\theta} \odot \boldsymbol{b}](\boldsymbol{y} \mid \boldsymbol{x}).$$

This can be approximated by scaling weights by their probability of being active.

Normalization

Goal: Make all units more similar. A unit $f : \mathbb{R}^d \to \mathbb{R}$ can be normalized by

$$\bar{f} = \frac{f - \mathbb{E}[f(\boldsymbol{x})]}{\sqrt{\mathrm{Var}[f(\boldsymbol{x})]}}.$$

This removes 2 DOF (bias and variance) → Explicitly parameterize:

$$\overline{f}[\mu, \gamma](\boldsymbol{x}) = \mu + \gamma \bar{f}(\boldsymbol{x}).$$

BatchNorm approximates $\mathbb{E}[f]$ and $\mathrm{Var}[f]$ over a mini-batch.

Normalization is very effective and sometimes essential. We used to believe that it was because it helped combat covariance shift. However, a modern motivation shows that normalization is the same as weight normalization and scaling by $\|\boldsymbol{w}\|_{\boldsymbol{I}}/\|\boldsymbol{w}\|_{\boldsymbol{\Sigma}}$, where $\boldsymbol{\Sigma} = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$.

Layer normalization estimates mean and variance over the feature dimension instead of batch dimension.

Weight normalization

Normalize weights before applying them:

$$f[\boldsymbol{v}, \gamma](\boldsymbol{x}) = \phi(\boldsymbol{w}^\top \boldsymbol{x}), \quad \boldsymbol{w} = \frac{\gamma}{\|\boldsymbol{v}\|_2}\boldsymbol{v}.$$

Gradients:

$$\frac{\partial h}{\partial \gamma} = \frac{\partial h}{\partial \boldsymbol{w}}\frac{\partial \boldsymbol{w}}{\partial \gamma}, \quad \frac{\partial h}{\partial \boldsymbol{v}} = \frac{\gamma}{\|\boldsymbol{v}\|}\frac{\partial h}{\partial \boldsymbol{w}}\left(\boldsymbol{I} - \frac{\boldsymbol{w}\boldsymbol{w}^\top}{\|\boldsymbol{w}\|^2}\right).$$

Here $\boldsymbol{I} - \boldsymbol{w}\boldsymbol{w}^\top/\|\boldsymbol{w}\|^2$ is the projection matrix onto the complement of $\boldsymbol{w}$ → The direction of $\boldsymbol{w}$ is projected out.

Data augmentation

Transform input data to train invariances.

Label smoothing: Replace labels by noisy probability distributions, because classifiers are not good at dealing with mislabeled data.

Distillation

Let F be the teacher and G its student and we want the student to match its teacher's logits. Then, we have tempered cross-entropy loss:

$$\ell(\boldsymbol{x}) = \sum_{y \in \mathcal{Y}} \frac{\exp(^{F_y(\boldsymbol{x})}/_T)}{\sum_{y' \in \mathcal{Y}} \exp(^{F_{y'}(\boldsymbol{x})}/_T)} \cdot \left(\frac{1}{T} G_y(\boldsymbol{x}) - \log \sum_{y' \in \mathcal{Y}} \exp(^{G_{y'}(\boldsymbol{x})}/_T)\right).$$

Gradient:

$$\frac{\partial h}{\partial G_y} = \frac{1}{T}\left(\frac{\exp(^{F_y(\boldsymbol{x})}/_T)}{\sum_{y' \in \mathcal{Y}} \exp(^{F_{y'}(\boldsymbol{x})}/_T)} - \frac{\exp(^{G_y(\boldsymbol{x})}/_T)}{\sum_{y' \in \mathcal{Y}} \exp(^{G_{y'}(\boldsymbol{x})}/_T)}\right).$$

Neural tangent kernel

Linearized model

$$f[\boldsymbol{\theta}] \approx f[\boldsymbol{\theta}_0] + \langle \nabla f[\boldsymbol{\theta}_0], \boldsymbol{\theta} - \boldsymbol{\theta}_0 \rangle.$$

f is non-linear w.r.t. $\boldsymbol{x}$, but linear w.r.t. $\boldsymbol{\theta}$ → Define linear model with gradient feature map:

$$h[\boldsymbol{\beta}](\boldsymbol{x}) = f[\boldsymbol{\theta}_0](\boldsymbol{x}) + \boldsymbol{\beta}^\top \nabla f[\boldsymbol{\theta}_0](\boldsymbol{x}).$$

Kernel method with $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \nabla f[\boldsymbol{\theta}_0](\boldsymbol{x}), \nabla f[\boldsymbol{\theta}_0](\boldsymbol{x}') \rangle$ and MSE loss,

$$\boldsymbol{\beta}^\star = \boldsymbol{\Phi}^\top \boldsymbol{K}^{-1}(\boldsymbol{y} - \boldsymbol{f}), \quad \boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top, \quad \boldsymbol{\phi}_i = \nabla f[\boldsymbol{\theta}_0](\boldsymbol{x}_i).$$

Predictions:

$$h^\star(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^\top \boldsymbol{K}^{-1}(\boldsymbol{y} - \boldsymbol{f}).$$

Linearized models are non-competitive with full networks. Also they may be intractable due to number of samples and parameters. Benefit: We can look at DNNs through the lens of kernel methods if the parameters do not evolve far away from $\boldsymbol{\theta}_0$.

Training dynamics

Gradient flow ODE with MSE loss:

$$\dot{\boldsymbol{\theta}}_t = \sum_{i=1}^{n} (y_i - f[\boldsymbol{\theta}_t](\boldsymbol{x}_i)) \nabla f[\boldsymbol{\theta}_t](\boldsymbol{x}).$$

Functional gradient flow:

$$\dot{f}_j[\boldsymbol{\theta}_t] = \dot{\boldsymbol{\theta}}_t{}^\top \nabla f[\boldsymbol{\theta}_t](\boldsymbol{x}_j) = \sum_{i=1}^{n} (y_i - f[\boldsymbol{\theta}_t](\boldsymbol{x}_i)) k[\boldsymbol{\theta}_t](\boldsymbol{x}_i, \boldsymbol{x}_j).$$

In matrix form:

$$\dot{\boldsymbol{f}}[\boldsymbol{\theta}_t] = \boldsymbol{K}[\boldsymbol{\theta}_t](\boldsymbol{y} - \boldsymbol{f}[\boldsymbol{\theta}_t]), \quad \dot{\boldsymbol{f}}[\boldsymbol{\theta}_t] = -\boldsymbol{K}[\boldsymbol{\theta}_t] \nabla_{\boldsymbol{f}[\boldsymbol{\theta}]} \ell(\boldsymbol{\theta}_t).$$

NTK $\boldsymbol{K}[\boldsymbol{\theta}_t]$ governs the evolution of the joint sample predictions. Problem: NTK has a dependence on the parameters.

Infinite width

In practice it has been found that as the width m of a model is scaled, the parameters stay more closely to their initialization during gradient descent. It can be shown (under basic assumptions) that the NTK converges in probability to a deterministic limit as the model is scaled to infinite width:

$$k[\boldsymbol{\theta}] \xrightarrow{m \to \infty} k_\infty.$$

The deterministic limit depends only on the law of initialization. Under these training dynamics, minimizing MSE equates to solving a kernel regression problem with k_∞. This provides insight into why overparameterization works so well in practice, despite having the ability to overfit.

NTK constancy

The NTK remains constant under gradient flow: $\frac{\partial \boldsymbol{K}[\boldsymbol{\theta}_t]}{\partial t} = \boldsymbol{0}$.

Bayesian learning

Goal is to compute the Bayesian predictive posterior:

$$f(\boldsymbol{x}) = \int p(\boldsymbol{\theta} \mid \mathcal{S}) f[\boldsymbol{\theta}](\boldsymbol{x}) \mathrm{d}\boldsymbol{\theta},$$

where

$$p(\boldsymbol{\theta} \mid \mathcal{S}) = \frac{p(\mathcal{S} \mid \boldsymbol{\theta})}{p(\mathcal{S})}, \quad p(\mathcal{S}) = \int p(\boldsymbol{\theta}) p(\mathcal{S} \mid \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}.$$

$p(\mathcal{S})$ is intractable but often not necessary. An isotropic Gaussian prior leads to MAP optimizing ℓ_2 regularization.

Markov chain Monte Carlo

However, we do not want MAP only, we want the full distribution. MCMC methods sample from the posterior by constructing a Markov chain, where the stationary distribution is the posterior.

Detailed Balance Equation:

$$q(\boldsymbol{\theta})\Pi(\boldsymbol{\theta}' \mid \boldsymbol{\theta}) = q(\boldsymbol{\theta}')\Pi(\boldsymbol{\theta} \mid \boldsymbol{\theta}'), \quad \forall \boldsymbol{\theta}, \boldsymbol{\theta}'.$$

Then, the Markov chain is time reversible and has the unique stationary distribution q.

Metropolis-Hastings samples from an arbitrary Markov chain with kernel $\tilde{\Pi}$ and adjusts it such that DBE is satisfied for the posterior. It does so by constructing a new kernel:

$$\Pi(\boldsymbol{\theta}' \mid \boldsymbol{\theta}) = \tilde{\Pi}(\boldsymbol{\theta}' \mid \boldsymbol{\theta})\alpha(\boldsymbol{\theta}' \mid \boldsymbol{\theta})$$

$$\alpha(\boldsymbol{\theta}' \mid \boldsymbol{\theta}) = \min\left\{1, \frac{p(\boldsymbol{\theta} \mid \mathcal{S})\tilde{\Pi}(\boldsymbol{\theta}' \mid \boldsymbol{\theta})}{p(\boldsymbol{\theta}' \mid \mathcal{S})\tilde{\Pi}(\boldsymbol{\theta} \mid \boldsymbol{\theta}')}\right\}.$$

This is the unique choice of acceptance function α that has a one-sided structure. If $\tilde{\Pi}$ is symmetric, we only need the ratio of posteriors, not $p(\mathcal{S})$. Problems: Burn-in period can be arbitrarily long due to poor $\tilde{\Pi}$ leading to high rejection probabilities.

Hamiltonian Monte Carlo obtains posterior averages. Energy function:

$$E(\boldsymbol{\theta}) = -\sum_{\boldsymbol{x}, y} \log p[\boldsymbol{\theta}](y \mid \boldsymbol{x}) - \log p(\boldsymbol{\theta}).$$

The Hamiltonian augments with momentum vector:

$$H(\boldsymbol{\theta}, \boldsymbol{v}) = E(\boldsymbol{\theta}) + \frac{1}{2}\boldsymbol{v}^\top \boldsymbol{M}^{-1}\boldsymbol{v}.$$

Hamiltonian dynamics:

$$\dot{\boldsymbol{v}} = -\nabla E(\boldsymbol{\theta}), \quad \dot{\boldsymbol{\theta}} = \boldsymbol{v}.$$

HMC discretizes with stepsize η:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \boldsymbol{v}_t, \quad \boldsymbol{v}_{t+1} = \boldsymbol{v}_t - \eta \nabla E(\boldsymbol{\theta}_t).$$

We sample the posterior by following momentum-based gradient descent dynamics. (We can also view this as momentum-based gradient descent leading to a single sample approximation of the predictive distribution.) Problem: We need to compute the full gradient.

Langevin dynamics extends HMC by friction:

$$\dot{\boldsymbol{\theta}} = \boldsymbol{v}, \quad \mathrm{d}\boldsymbol{v} = -\nabla E(\boldsymbol{\theta})\mathrm{d}t - \boldsymbol{B}\boldsymbol{v}\mathrm{d}t + \mathcal{N}(\boldsymbol{0}, 2\boldsymbol{B}\mathrm{d}t).$$

Friction reduces momentum and dissipates kinetic energy, while the Wiener noise injects stochasticity. Discretize:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \boldsymbol{v}_t$$

$$\boldsymbol{v}_{t+1} = (1 - \eta\gamma)\boldsymbol{v}_t - \eta s \nabla \tilde{E}(\boldsymbol{\theta}) + \sqrt{2\gamma\eta}\mathcal{N}(\boldsymbol{0}, \boldsymbol{I}).$$

Here, $\tilde{E}$ is a stochastic potential function which is the empirical loss over a random mini-batch of data.

Gaussian process

GPs is a fully tractable Bayesian method. f is a GP if for every finite subset $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathcal{X}$, the resulting finite marginal is jointly normally distributed,

$$[f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)] \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}).$$

In GPs, the mean can be computed by deterministic regression and the covariance matrix is evaluated by a kernel function:

$$\sigma_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j).$$

The kernel function can be seen as a prior over function space that describes how related the output values corresponding to the two input value should be. RBF kernel encodes that close input value should have close output values:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma\|\boldsymbol{x} - \boldsymbol{x}'\|^2).$$

Linear networks assume a random Gaussian weight vector:

$$\boldsymbol{w} \sim \mathcal{N}\left(\boldsymbol{0}, \frac{\sigma^2}{d}\boldsymbol{I}_d\right).$$

Outputs are computed by $y_i = \boldsymbol{w}^\top \boldsymbol{x}_i$. Vectorized:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}.$$

This is a Gaussian vector:

$$\boldsymbol{y} \sim \mathcal{N}\left(\boldsymbol{0}, \frac{\sigma^2}{d}\boldsymbol{X}^\top \boldsymbol{X}\right).$$

In other words, it is a GP with the following kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \frac{\sigma^2}{d}\boldsymbol{x}^\top \boldsymbol{x}'.$$

We can do this for multiple units because the preactivations of units in the same layer are independent, conditioned on the input. In general, we do not get the same effect if we increase the depth, because there is randomness not only in the weights but also the preactivations. However, a deep preactivation process is "near normal" for high-dimensional inputs.

We can extend this to non-linear networks. But, the activations are no longer Gaussian due to the non-linearity. However, due to CLT, they are effectively shaped back into Gaussians when they propagate to the next layer. The mean function and kernels are computed by

$$\mu(\boldsymbol{x}^\ell) = \mathbb{E}[\phi(\boldsymbol{W}^{\ell-1}\boldsymbol{x}^{\ell-1})], \quad k^\ell(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathbb{E}[\phi(\boldsymbol{x}_i^{\ell-1})\phi(\boldsymbol{x}_j^{\ell-1})].$$

We can now use kernel regression:

$$f^\star(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^\top \boldsymbol{K}^{-1}\boldsymbol{y}, \quad k = k^L.$$

In conclusion, DNNs in the infinite-width limit can be thought of as GPs (because then all preactivations can be viewed as Gaussians). Benefit: Uncertainty quantification, No training, Leverage tricks form kernel machines. Problems: Computing $f^\star$ and storing $\boldsymbol{K}^\ell$ are not feasible, It is much less efficient than optimizing weights with gradient descent.

Statistical learning theory

VC theory

Let $\mathcal{F}$ be a class of binary classifiers. Then the following is the set of possible classification outcomes over a dataset $\mathcal{S}$:

$$\mathcal{F}(\mathcal{S}) = \{[f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)] \in \{0,1\}^n \mid f \in \mathcal{F}\}.$$

Further define maximum number:

$$\mathcal{F}(n) = \sup_{|\mathcal{S}|=n} |\mathcal{F}(\mathcal{S})|.$$

$\mathcal{F}$ shatters $\mathcal{S}$ if $|\mathcal{F}(\mathcal{S})| = 2^n$ (every possible labeling is realized by some function $f \in \mathcal{F}$). VC dimensionality is defined as

$$\mathrm{VC}(\mathcal{F}) = \max_{n \in \mathbb{N}} \sup_{|\mathcal{S}|=n} \mathbb{1}\{\mathcal{F}(n) = 2^n\}.$$

Under uniform convergence, VC inequality holds:

$$\mathbb{P}\left(\sup_{f \in \mathcal{F}} |\hat{\ell}(f) - \ell(f)| > \epsilon\right) \leq 8|\mathcal{F}(n)| \exp\left(-\frac{n\epsilon^2}{32}\right),$$

where ℓ is the expected loss and $\hat{\ell}$ is the empirical loss. Intuition: No generalizes guarantees can be given if $\mathcal{F}$ can be fit to any labeling.

Randomization experiments with CIFAR-10 observations:

1. DNNs can perfectly fit the training data;
2. When randomly replacing training labels, the models can still perfectly fit the data (memorization);
3. The training time does not increase much when labels are randomized;
4. When randomly shuffling pixels, the models can also perfectly fit the data → Inductive bias of CNNs does not provide much benefit in this regard.

These findings are unexplainable by the above theory.

Overparameterization can lead to double descent phenomenon, where large models will eventually start generalizing better after overfitting.

The flatness of local minima are linked to their generalization ability, because at flat minima, small perturbations in the parameters will only have a small effect on performance. These can be found by small-batch SGD, weight averaging, or entropy SGD.

PAC-Bayesian

For any $p \gg q$ and p-measurable X,

$$\mathbb{E}_q[X] \leq D_{\mathrm{KL}}(q\|p) + \log \mathbb{E}_p[\exp(X)].$$

For a fixed p, any q, and $\epsilon \in (0,1)$, we have the following with probability greater than or equal to ϵ,

$$\mathbb{E}_q[\ell(f)] - \mathbb{E}_q[\hat{\ell}(f)] \leq \sqrt{\frac{2}{n}\left(D_{\mathrm{KL}}(q\|p) + \log\frac{2\sqrt{n}}{\epsilon}\right)}.$$

Here, p is a prior over parameters, q the posterior, and we bound the expected generalization gap over stochastic classifiers → $\mathcal{O}(1/\sqrt{n})$ bound on generalization error. However, it only applies to stochastic classifiers, not single classifiers.

This motivated the PAC Bayes loss function

$$\ell_{\mathrm{PAC}}(q) \doteq \mathbb{E}_q[\hat{\ell}] + \sqrt{\frac{2}{n}\left(D_{\mathrm{KL}}(q\|p) + \log\frac{2\sqrt{n}}{\epsilon}\right)}.$$

Effectively just a regularization term. The KL term can be computed in closed form if prior and posterior are Gaussian.

Generative models

Autoencoders

Encoder $\boldsymbol{\mathcal{E}}$ maps data to latents and decoder $\boldsymbol{\mathcal{D}}$ maps latents to data. We want $\boldsymbol{\mathcal{D}}(\boldsymbol{\mathcal{E}}(\boldsymbol{x})) = \boldsymbol{x}$. Linear autoencoder with MSE loss is PCA of covariance matrix $\frac{1}{n}\boldsymbol{X}^\top \boldsymbol{X}$ and taking m principal eigenvectors. Intuition: Retain as much variance as possible.

VAE optimizes log-likelihood of data → Intractable → ELBO:

$$\log p[\boldsymbol{\theta}](\boldsymbol{x}) \geq \mathbb{E}_{p[\boldsymbol{\vartheta}](\boldsymbol{z}|\boldsymbol{x})}[\log p[\boldsymbol{\theta}](\boldsymbol{x} \mid \boldsymbol{z})] - D_{\mathrm{KL}}(p[\boldsymbol{\vartheta}](\boldsymbol{z} \mid \boldsymbol{x})\|p(\boldsymbol{z})).$$

$p[\boldsymbol{\vartheta}]$ is the encoder distribution and $p[\boldsymbol{\theta}]$ is the decoder distribution. This is effectively a reconstruction loss with a regularization term, where the regularization term ensures a well-behaved latent space.

In general, the posterior $p[\boldsymbol{\vartheta}](\boldsymbol{z} \mid \boldsymbol{x})$ is intractable, so we restrict it to Gaussians,

$$\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\vartheta} \sim \mathcal{N}(\boldsymbol{\mu}[\boldsymbol{\vartheta}](\boldsymbol{x}), \boldsymbol{\Sigma}[\boldsymbol{\vartheta}](\boldsymbol{x})).$$

And the prior is $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Then, the KL divergence can be computed in a closed form:

$$\frac{1}{2}\left(\|\boldsymbol{\mu}[\boldsymbol{\vartheta}](\boldsymbol{x})\|^2 + \mathrm{tr}(\boldsymbol{\Sigma}[\boldsymbol{\vartheta}](\boldsymbol{x})) - \log\det(\boldsymbol{\Sigma}[\boldsymbol{\vartheta}](\boldsymbol{x})) - m\right).$$

This can be optimized using the reparameterization tick.

Generative adversarial networks

Log-likelihood is not the only way to optimize a model. GANs provide a training signal by introducing a binary classifier that distinguishes between samples from "nature" (1) and the generator (0): discriminator.

Augmented distribution over samples:

$$\tilde{p}(\boldsymbol{x}, y) = \frac{1}{2}(yp(\boldsymbol{x}) + (1-y)p[\boldsymbol{\theta}](\boldsymbol{x})).$$

Bayes-optimal classifier:

$$\mathbb{P}(y = 1 \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x})}{p(\boldsymbol{x}) + p[\boldsymbol{\theta}](\boldsymbol{x})}.$$

Minimizing the logistic log-likelihood w.r.t. this discriminator gives the following loss for the generator,

$$\ell^\star(\boldsymbol{\theta}) = D_{\text{JS}}(p \| p[\boldsymbol{\theta}]) - \log 2,$$

where $D_{\text{JS}}(p\|q) = H(^{p+p[\boldsymbol{\theta}]}/_2) - {}^{H(p)+H(p[\boldsymbol{\theta}])}/_2$. But, the optimal classifier is intractable, so we train a parametrized one $q[\boldsymbol{\varphi}]$,

$$\boldsymbol{\theta}^\star, \boldsymbol{\varphi}^\star \in \operatorname*{argmin}_{\boldsymbol{\theta}} \operatorname*{argmax}_{\boldsymbol{\varphi}} \ell(\boldsymbol{\theta}, \boldsymbol{\varphi}),$$

where

$$\ell(\boldsymbol{\theta}, \boldsymbol{\varphi}) = \mathbb{E}_{\tilde{p}[\boldsymbol{\theta}]}[y \log q[\boldsymbol{\varphi}](\boldsymbol{x}) + (1-y)\log(1 - q[\boldsymbol{\varphi}](\boldsymbol{x}))].$$

We have the following bound:

$$\ell^\star(\boldsymbol{\theta}) \geq \sup_{\boldsymbol{\varphi}} \ell(\boldsymbol{\theta}, \boldsymbol{\varphi}).$$

Problem: Gradient descent-ascent is not guaranteed to converge. Solution: Extragradient optimization algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}_{t+1/2}, \boldsymbol{\varphi}_t), \quad \boldsymbol{\theta}_{t+1/2} = \boldsymbol{\theta}_t - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t)$$

$$\boldsymbol{\varphi}_{t+1} = \boldsymbol{\varphi}_t + \eta \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \ell(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_{t+1/2}), \quad \boldsymbol{\varphi}_{t+1/2} = \boldsymbol{\varphi}_t + \eta \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \ell(\boldsymbol{\theta}_t, \boldsymbol{\varphi}_t).$$

In practice, we also need to use a different loss function for the generator:

$$\ell(\boldsymbol{\theta} \mid \boldsymbol{\varphi}) = \mathbb{E}_{p[\boldsymbol{\theta}]}[-\log q[\boldsymbol{\varphi}](\boldsymbol{x})],$$

because otherwise the gradient goes to infinity when $q[\boldsymbol{\varphi}](\boldsymbol{x}) = 1$, which makes the generator saturate.

Diffusion models

Map a simple distribution to a complex one in many steps:

$$\pi = \pi_T \mapsto \pi_{T-1} \mapsto \cdots \mapsto \pi_0 \approx p.$$

SDE view:

$$\mathrm{d}\boldsymbol{x}_t = -\frac{1}{2}\beta_t \boldsymbol{x}_t \mathrm{d}t \sqrt{\beta_t} \mathrm{d}\omega_t.$$

Time-reversed:

$$\mathrm{d}\boldsymbol{x}_t = \left(-\frac{1}{2}\beta_t \boldsymbol{x}_t - \beta_t \boldsymbol{\nabla}_{\boldsymbol{x}_t} \log q_t(\boldsymbol{x}_t)\right)\mathrm{d}t + \sqrt{\beta_t}\mathrm{d}\bar{\omega}_t.$$

Denoising amounts to approximating a vector field over the gradient of the probability distribution moving towards areas with high probability density. Score models approximate $\boldsymbol{\nabla}_{\boldsymbol{x}_t} \log q_t(\boldsymbol{x}_t)$.

ELBO view: Forward process:

$$\boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}).$$

Energy of the stochastic process evolves as $\mathbb{E}[\|\boldsymbol{x}_t\|^2 \mid \boldsymbol{x}_{t-1}] = (1-\beta_t)\|\boldsymbol{x}_{t-1}\|^2 + \beta_t \mathrm{tr}(\boldsymbol{I})$. If $\mathbb{E}[\|\boldsymbol{x}_0\|^2] = \mathrm{tr}(\boldsymbol{I}) = \dim(\boldsymbol{x}_0)$, then energy is conserved throughout the process.

Closed form ($\bar{\alpha}_t = \prod_{\tau=1}^t (1-\beta_\tau)$):

$$\boldsymbol{x}_t \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}, (1-\bar{\alpha}_t)\boldsymbol{I}).$$

Backward process:

$$\boldsymbol{x}_{t-1} \sim \mathcal{N}(\boldsymbol{\mu}[\boldsymbol{\theta}](\boldsymbol{x}_t, t), \boldsymbol{\Sigma}[\boldsymbol{\theta}](\boldsymbol{x}_t, t)).$$

ELBO:

$$\log p[\boldsymbol{\theta}](\boldsymbol{x}_0) \geq \sum_{t=0}^T \ell_t,$$

where

$$\ell_t = \begin{cases} \mathbb{E}_q[\log p[\boldsymbol{\theta}](\boldsymbol{x}_0 \mid \boldsymbol{x}_1)] & t = 0 \\ -D_{\text{KL}}(q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_0) \| p[\boldsymbol{\theta}](\boldsymbol{x}_t \mid \boldsymbol{x}_{t+1})) & 0 < t < T \\ -D_{\text{KL}}(q(\boldsymbol{x}_T \mid \boldsymbol{x}_0) \| \pi) & t = T. \end{cases}$$

The KL divergences can be analytically computed because all q_t are Gaussians an we parameterized the network as a Gaussian. The q targets are derived as

$$q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}_t, \boldsymbol{x}_0, t), \tilde{\beta}_t),$$

where

$$\boldsymbol{\mu}(\boldsymbol{x}_t, \boldsymbol{x}_0, t) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}_0 + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\sqrt{1-\beta_t}\boldsymbol{x}_t$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t.$$

Thus ℓ_t simplify to

$$\ell_t = -\frac{1}{2\sigma_t^2}\|\boldsymbol{\mu}(\boldsymbol{x}_t, \boldsymbol{x}_0, t) - \boldsymbol{\mu}[\boldsymbol{\theta}](\boldsymbol{x}_t, t)\|^2,$$

where $\sigma_t^2 \in [\beta_t, \tilde{\beta}_t]$.

By noting the closed form forward process, we have

$$\boldsymbol{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\boldsymbol{x}_t - \frac{\sqrt{1-\bar{\alpha}_t}}{\bar{\alpha}_t}\boldsymbol{\epsilon}.$$

We can thus rewrite

$$\boldsymbol{\mu}(\boldsymbol{x}_t, \boldsymbol{x}_0, t) = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}\right).$$

Note that $\boldsymbol{\epsilon}$ fully determines $\boldsymbol{x}_t$ and $\boldsymbol{x}_0$ is constant. So, we only need to predict $\boldsymbol{\epsilon}$. Simplified loss:

$$\mathbb{E}_q[\ell_t \mid \boldsymbol{x}_0] = \mathbb{E}_{\boldsymbol{\epsilon}}[\lambda(t)\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}[\boldsymbol{\theta}](\boldsymbol{x}_t, t)\|^2]$$

$$\lambda(t) = \frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)}.$$

In practice, the loss is approximated by

$$\ell(\boldsymbol{\theta} \mid \boldsymbol{x}_0) = \frac{1}{T}\sum_{t=1}^T [\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}[\boldsymbol{\theta}](\boldsymbol{x}_t, t)\|^2 \mid \boldsymbol{x}_0].$$

Entropy bound:

$$H(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) \leq H(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}).$$

The entropy of the reverse process is bounded by the entropy of the forward process.

Adversarial attacks

The attacker wants to make small changes to the input such that the model gives a different result.

p-norm robustness

Consider a multi-class classifier $f : \mathbb{R}^d \to [m]$. The goal of an adversarial attack is to find a perturbation $\boldsymbol{\eta}$ such that

$$f(\boldsymbol{x} + \boldsymbol{\eta}) \neq f(\boldsymbol{x}), \quad \|\boldsymbol{\eta}\|_p \leq \epsilon.$$

Consider a binary affine classifier and $p = 2$,

$$f(\boldsymbol{x}) = \operatorname{argmax}\{\boldsymbol{w}_1^\top \boldsymbol{x} + b_1, \boldsymbol{w}_2^\top \boldsymbol{x} + b_2\}.$$

Assume $\boldsymbol{x}$ is classified as 1 and we want to find $\boldsymbol{\eta}$ such that $\boldsymbol{x} + \boldsymbol{\eta}$ is classified as 2 such that $\|\boldsymbol{\eta}\|_2$ is minimized $\to$ Convex program:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{\eta}\|_2^2 \\ \text{subject to} \quad & (\boldsymbol{w}_1 - \boldsymbol{w}_2)^\top(\boldsymbol{x} + \boldsymbol{\eta}) + b_1 - b_2 \leq 0. \end{aligned}$$

Set gradient of Lagrangian to zero $\to \boldsymbol{\eta}^\star = \lambda(\boldsymbol{w}_2 - \boldsymbol{w}_1)$. Then, find λ that satisfies constraint $\to \lambda \geq {}^{f_1(\boldsymbol{x}) - f_2(\boldsymbol{x})}/_{\|\boldsymbol{w}_1 - \boldsymbol{w}_2\|_2^2}$. Thus:

$$\boldsymbol{\eta}^\star = \frac{f_1(\boldsymbol{x}) - f_2(\boldsymbol{x})}{\|\boldsymbol{w}_2 - \boldsymbol{w}_1\|_2^2}(\boldsymbol{w}_2 - \boldsymbol{w}_1).$$

This can be generalized to any source i and target j. In the general case, we can linearize the model and iteratively solve the above convex program.

Robust training

Robust training systematically makes models robust to adversarial attacks by extending the loss function to neighborhoods of training points:

$$\ell(\boldsymbol{x}) \mapsto \max_{\boldsymbol{\eta}: \|\boldsymbol{\eta}\|_p \leq \epsilon} \ell(\boldsymbol{x} + \boldsymbol{\eta}).$$

This can be solved with projected gradient ascent. For $p = 2$, we have

$$\boldsymbol{\eta}_{t+1} = \epsilon \Pi(\boldsymbol{\eta}_t + \alpha \boldsymbol{\nabla}_{\boldsymbol{x}} \ell(\boldsymbol{x} + \boldsymbol{\eta}_t)), \quad \Pi(\boldsymbol{z}) \doteq \frac{\boldsymbol{z}}{\|\boldsymbol{z}\|_2}.$$

Fast gradient sign method performs one iteration with $p = \infty$ resulting in $\boldsymbol{\eta} = \epsilon \cdot \operatorname{sgn}(\boldsymbol{\nabla}_{\boldsymbol{x}} \ell(\boldsymbol{x}))$