- PERCEPTRONS (3)
- CLASS, PAC, VC (12)
- INFER, log-likel., cross-entr. (4)
- MLP (6)
- GRAD. DESCENT (15)
- AUTO. DIFF (12)
- REGULARIZATION (13)
- CNN (12)
- ATTENTION, NLP (28)
- RL (8)

PERCEPTRONS

- McCulloh, Pitts 43'
- FROM LOGIC GATES TO CONNECTIONIST MODEL OF BRAIN (IMAGE RECOGNITION)
- $\bar{x} \in \mathbb{R}^N$ is INPUT
- $\bar{w} \in \mathbb{R}^N$ WEIGHTS
- ϑ THRESHOLD (OR BIAS: $b = -\vartheta$)
- $\hat{y}_{\bar{w},b}(\bar{x}) = \text{sign}(\bar{w}\cdot\bar{x} + b)$
- $\bar{w}, \vartheta$ IS THE FULL MODEL
- NOT BIOLOGICAL:
  - → NO TIME VARIABLE (INSTANT RESPONSES)
  - → UNCONSTRAINED INPUT (INSTEAD OF $\mathbb{R}^N_+$)
  - → SUMMATION OF INPUTS IS LINEAR AS IN DOT PRODUCT

BINARY CLASSIFIER: $(f : \mathbb{R}^N \to \{\pm 1\})$

DIVIDES $\mathbb{R}^N$ IN TWO USING HYPERPLANE WHOSE NORMAL IS $\bar{w}$ AND b IS DISTANCE FROM THE ORIGIN (MODEL IS INVARIANT TO $c\cdot(\bar{w}, b)$ SCALING)

P. LEARNING ALG (UPDATE WEIGHTS USING FEEDBACK)

$S = \{(\bar{x}^\mu, y^\mu)\}_{\mu\in[m]}$ DATASET

$R_S(\bar{w}, b) = \frac{1}{m}\sum_\mu \mathbb{1}_{\hat{y}(\bar{x}^\mu)\neq y^\mu}$ EMPIRICAL RISK, OR $\Theta(\bar{S}\cdot\bar{y}^\mu)$

$\Theta(\lambda) = \begin{cases} 1 & \lambda > 0 \\ 0 & \lambda < 0 \end{cases}$

UPDATING NUDGES THE INPUT IN THE RIGHT DIRECTION

$R_S = 0 \Rightarrow$ STOP

PERFORMANCE IS NOT MONOT. INCREASING, BUT IF $\exists\, \bar{w}, b : R_S = 0$, IT WILL TAKE $\# < \infty$ TO CONVERGE

YOU NEED LINEAR SEPARABILITY, OTHERWISE IT KEEPS EXPLORING BAD STATES WITHOUT CONVERGENCE (XOR PROBLEM)

1. $(t=0)\ \{\bar{w} = \bar{0},\ b = 0$
2. IF μ MISCLASSIFIED, $\{\bar{w} = \bar{w} + y^\mu x^\mu$, $b = b + y^\mu$

INFERENCE

$S \sim iid$, MODEL $\mathcal{D}(y|\bar{x})$:
- $P : \mathcal{X} \to [0,1]^{|\mathcal{Y}|}$ (NORMALISED)
- $\mathcal{H} = \{P_\vartheta | \vartheta\}$
- $\exists \vartheta : \mathbb{P}(\mathcal{O} = \vartheta)$
- $\bar{x} \sim \mathcal{D}_X$
- $y \sim \bar{P}_\vartheta(\bar{x})$

PERCEPTRON: $\bar{P}^y_{\bar{\omega},b}(\bar{x}) = (1 + e^{-2y(\bar{\omega}\cdot\bar{x}+b)})^{-1}$

AKA. LOGISTIC REGRESSION (HERE, A GREATER NORM PRODUCES MORE PEAKED RESULTS)

CLASSIFIER:
- → STOCHASTIC (SAMPLE FROM DISTRIBUTION)
- → DETERMINISTIC (TAKE MLE)

ADDING BACK A PRIOR IS EASY, ADD $-\frac{1}{m}\log P(\vartheta)$ (m^{-1} ENSURES NORMALISATION)

BUT CHOICE OF PRIOR IS NOT OBVIOUS

BAYES-OPTIMAL CLASSIFIER IF THE CLASSIFIER HAS TO BE DETERMINISTIC, USE

$h_\vartheta(\bar{x}) = \text{argmax}_{y\in\mathcal{Y}} P^y_\vartheta(\bar{x})$

THIS IS BECAUSE THE MAXIMUM DISTRIBUTION IS CONCENTRATED AT THE MAXIMUM.

MAXIMUM A PRIORI:

$P(\vartheta|S) = \frac{P(S|\vartheta)\cdot P(\vartheta)}{P(S)}$

$\vartheta^* = \text{argmax}(\log P(S|\vartheta) + \log P(\vartheta))$ (S iid)

$= \log\prod_\mu P[(x^\mu, y^\mu)|\vartheta]$

$= \log\prod_\mu P[y^\mu|x^\mu,\vartheta]\cdot P[x^\mu|\vartheta]$ (NO ϑ DEPENDENCY)

$= \sum_\mu \log P^{y^\mu}_\vartheta(x^\mu) + \sum_\mu \log P_X(x^\mu)$

$\vartheta^* = \text{argmax}\left[-\sum_{\mu=1}^m H(\bar{q}^\mu, \bar{P}_\vartheta(\bar{x}^\mu)) + \log P(\vartheta)\right]$

CROSS-ENTROPY; FLAT PRIOR OVER LARGE REGION

JUST USE CROSS-ENTROPY LOSS → $\mathcal{L}^{XE_{nt}}_S(\vartheta) = \frac{1}{m}\sum_{\mu=1}^m H(\bar{q}^\mu, \bar{P}_\vartheta(\bar{x}^\mu))$

(BAYES-OPTIMAL CLASSIFIER MODEL) (≥ 0), EQUALS ZERO IFF COINCIDE

ENTROPY MAXIMISATION PRINCIPLE

ENSEMBLING: AVERAGE $\bar{P}(\bar{x})$ OF MORE MODELS (ALL WITH LOW $\mathcal{L}^{XE_{nt}}$) AND TAKE argmax → $h^{opt}_S(\bar{x}) = \text{argmax}\int d\vartheta\, P(\vartheta) e^{-m\mathcal{L}^{XE_{nt}}_S(\vartheta)} P^y_\vartheta(\bar{x})$

THEORETICALLY MINIMISES $\mathbb{E}[\text{ERROR}]$

CLASSIFICATION

$f : \mathcal{X} \to \mathcal{Y}$ (INPUT) (SET OF) (CLASSES)

ONE-HOT ENCODING: THE OUTPUS IS e_i IF THE LABEL IS i

BINARY LABELS REQUIRE ONLY ONE NUMBER (SPECIAL CASE)

SUPERVISED LEARNING
- TRAINING SET S IS R.V. FROM A DISTRIBUTION
- KNOWING FIRST m CASES, INFER μ_{m+1} LABEL:

$\mathbb{P}\left(y^{m+1} = y \,\middle|\, \substack{(x^\mu, y^\mu)_{\mu\in[m]} \\ x^{m+1} = x}\right)$

FREQUENTIST APPROACH:

$\forall \mu : (x^\mu, y^\mu) \overset{iid}{\sim} \mathcal{D}$, SO ONE CAN COMPUTE $\mathcal{D}_X(x)$, $\mathcal{D}(y|x)$

OR SIMPLY THE OPTIMAL DETERMINISTIC CLASSIFIER: $h : \mathcal{X} \to \mathcal{Y}$ (NO NEED TO MODEL $\mathcal{D}_X$)

PAC-LEARNING FORM

ASSUMPTION $\Rightarrow \forall \varepsilon, \delta \in (0,1)^2$, (GENERAL FORM) IF CONDITION, $\mathbb{P}(\text{ERROR} < \varepsilon) \geq 1-\delta$

- $\varepsilon \to$ ACCURACY
- $1-\delta \to$ CONFIDENCE

THE PURPOSE IS QUANTIFYING ERROR $\forall \varepsilon, \delta$

$\mathcal{H} = \{h \,|\, h : \mathcal{X} \to \mathcal{Y}\}$ HYPOTHESIS SPACE

(TOO SMALL → NOT RICH/DESCRIPTIVE ENOUGH | TOO LARGE → RISK OF OVERFIT)

PAC-LEARNING QUANTIFIES THESE TRADEOFFS DUE TO THE HYPOTHESIS SPACE

PAC-LEARNABLE

$X \sim \mathcal{D}_X(y)$, y IS ASSIGNED DETERMINISTICALLY BY NATURE

RISK (DETERMINISTIC) (AKA. LOSS → $\mathbb{P}$(ERROR))

$R_{\mathcal{D}_X, f}(h) = \mathbb{P}_{x\sim\mathcal{D}_X}[h(x) \neq f(x)] = \mathbb{E}_{x\sim\mathcal{D}_X}[\mathbb{1}_{h(x)\neq f(x)}]$

YOU WANT TO MINIMISE IT BUT DON'T HAVE ACCESS TO IT.

EMPIRICAL RISK (ER)

SAMPLE AVERAGE OF RISK

$R_S(h) = \frac{1}{m}\sum_{\mu\in[m]} \mathbb{1}_{h(x^\mu)\neq y^\mu}$

AN ERM ALGORITHM PICKS $h_S \in \mathcal{H}$ ST. ER IS MINIMISED

$\mathcal{A} : (\mathcal{X}\times\mathcal{Y})^m \to \mathcal{H}$

$\mathcal{D}_X, f$ REALIZABLE IN $\mathcal{H}$

$\exists h^* \in \mathcal{H}$ ST. $R_{\mathcal{D}_X,f}(h^*) = 0$

$(\text{UP TO } \mu.0 \Rightarrow R_S(h^*) = 0$ a.s.)

$\mathcal{H}$ IS PAC-LEARNABLE: → FOR EACH $\mathcal{D}_X, f$, THERE'S A WAY TO COMPUTE A m # OF SAMPLES ST. PAC HOLDS.

$\exists \mathcal{A} : (\mathcal{X}\times\mathcal{Y})^m \to \mathcal{H}$ ALGORITHM, $\exists m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ ST. $\forall \varepsilon, \delta \in (0,1)^2$, $\forall \mathcal{D}_X, \forall f : \mathcal{X}\to\mathcal{Y}$

FOR $\#$ SAMPLES, DESIRED ACCURACY, WHAT IS THE CONFIDENCE?

PAC-LEARNABILITY IS ABOUT HAVING ENOUGH SAMPLES TO INFER STRUCTURE

IF REALIZABLE, $m \geq m_{\mathcal{H}}(\varepsilon, \delta) \Rightarrow \mathbb{P}_{S\sim\mathcal{D}^m}[R_{\mathcal{D}_X,f}(\mathcal{A}(S)) \leq \varepsilon] \geq 1-\delta$ (THE $\mathcal{A}$ TAKES ONLY A m DATASET S)

$\mathcal{H}$ FINITE IS PAC-LEARNABLE → $\mathcal{A} = \text{ERM}_{\mathcal{H}}$, $m_{\mathcal{H}}(\varepsilon, \delta) = \left\lceil \frac{\log\frac{|\mathcal{H}|}{\delta}}{\varepsilon} \right\rceil$

COROLLARY: $\forall \varepsilon \in (0,1)$, $\forall m \in \mathbb{N}$, $\forall \mathcal{D}_X, \forall f : \mathcal{X}\to\mathcal{Y}$ — IF REALIZABILITY HOLDS, $\mathbb{P}_{S\sim\mathcal{D}^m}[R_{\mathcal{D}_X,f}(\text{ERM}_{\mathcal{H}}(S)) \leq \varepsilon] \geq 1 - |\mathcal{H}|e^{-\varepsilon m}$ (THE BOUND CAN BE VACUOUS, ADDING NOTHING)

HAVING A CONTINUOUS PARAMETER MAKES $|\mathcal{H}| = \infty$ BUT FLOATS USING n BITS HAS $|\mathcal{H}| \leq 2^n$, $\log|\mathcal{H}| \propto \#\text{PARAMS}$

$\mathcal{H}$ SHATTERS $C = (x^\mu)_{\mu\in[m]}$

$\forall (y^\mu)_{\mu\in[m]}$, $\exists h \in \mathcal{X} : \forall \mu,\ h(x^\mu) = y^\mu$

EVERY POSSIBLE LABELLING OF C IS REALIZABLE IN $\mathcal{H}$

NOTE: $O(m^n) = \#$ POSSIBLE LABELLINGS SO $\text{VC}_{\dim} = d$ IMPLIES $\exists m.n$ BOUNDARY.

VC-DIMENSION OF $\mathcal{H}$ IS THE MAX SIZE OF C SET THAT CAN BE SHATTERED BY $\mathcal{H}$:
- $\text{VC}_{\dim}$ IS LARGER THAN THE SIZE OF ANY SET THAT CAN BE SHATTERED BY $\mathcal{H}$
- PROVING $\text{VC}_{\dim}$ INVOLVES FINDING THE UPPER BOUND ON SHATTERED SETS

PAC-LEARNABLE $\mathcal{H}$ CLASS **IFF** VC-DIMENSION IS FINITE. THEN ANY ERM ALG IS PAC:

$m_{\mathcal{H}}(\varepsilon, \delta) \leq h. \frac{d.\log(\varepsilon) + \log(\delta)}{\varepsilon}$

PERCEPTRONS ON N-DIM. DATA HAVE $\text{VC}_{\dim} = N+1$

$\text{VC}_{\dim}$ MEASURES EXPRESSIVITY OF $\mathcal{H}$ IF $\text{VC} = d$, YOU CAN'T FIT SETS OF ARBITRARY SIZE.

ASSUME LABELS FOLLOW $y \sim \mathcal{D}(y|x)$ (MAYBE IF DATA CANNOT PROVIDE ENOUGH INFORMATION) ↑ INCOMPLETE DATA

RISK $R_{\mathcal{D}}(h) = \mathbb{P}_{x,y\sim\mathcal{D}}[h(x) \neq y]$ (REDUCES TO PREVIOUS CASE IF DETERMINISTIC)

BAYES-OPTIMAL CLASSIFIER

$\mathcal{D}(x,y) = \mathcal{D}_X(x)\mathcal{D}(y|x)$

$f_{\mathcal{D}}(x) = \text{argmax}_{y\in\mathcal{Y}} \mathcal{D}(y|x)$

THEN THE **INTRINSIC RISK:** $\varepsilon_{\text{int}} = R_{\mathcal{D}}(f_{\mathcal{D}}) = \mathbb{E}_{x\sim\mathcal{D}_X}[1 - \max_y \mathcal{D}(y|x)]$ ($=0$ IFF DETERMINISTIC)

SERVES AS LOWER BOUND TO $\mathcal{H}$.

$\varepsilon_{\text{app}} = R_{\mathcal{D}}(h_{\mathcal{H},\mathcal{D}}) - R_{\mathcal{D}}(f_{\mathcal{D}}) \geq 0$ (ADDITIONAL ERROR FROM CHOICE OF $\mathcal{H}$)

$R_{\mathcal{D}}(h_S) = \varepsilon_{\text{int}} + \varepsilon_{\text{app}} + \varepsilon_{\text{est}}$ → DECREASE IN m,
- ε_{int}: CAN'T DO ANYTHING
- ε_{app}: BIAS FROM CHOICE OF $\mathcal{H}$ INCREASE IN $\text{VC}_{\dim}(\mathcal{H})$ (OVERFITTING)
- ε_{est}: ESTIMATION, DEPENDS ON $\#$ SAMPLES

$\mathcal{H}$ IS AGNOSTIC-PAC-LEARNABLE:

$\exists \mathcal{A} : (\mathcal{X}, \mathcal{Y})^m \to \mathcal{H}$

$\exists m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ ST. $\forall \delta, \varepsilon \in (0,1)^2$

$\forall \mathcal{D}$ $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$

$\mathbb{P}_{S\sim\mathcal{D}^m}[\underbrace{R_{\mathcal{D}}(\mathcal{A}(S)) - R_{\mathcal{D}}(h_{\mathcal{H},\mathcal{D}})}_{\varepsilon_{\text{est}}} \leq \varepsilon] \geq 1-\delta$

→ $\mathcal{H}$ IS NOT SO LARGE IT OVERFITS.

$\mathcal{H}$ IS AGN-PAC-LEARN **IFF** $\text{VC}_{\dim}(\mathcal{H}) = d < \infty$

ANY ERM IS SUCCESSFUL. $m_{\mathcal{H}}(\varepsilon, \delta) \in h. \frac{d - \log(\delta)}{\varepsilon^2}$

REDUCE INDUCTIVE BIAS: KNOWING m, GOAL ε, δ PICK $\mathcal{H}$ ST. $\text{VC}_{\dim} < d_{\max}(m, \varepsilon, \delta)$

MULTILAYER PERCEPTRON (MLP)

UNIVERSAL APPROXIMATION THM:
- $K \subset \mathbb{R}^N$ COMPACT
- $g : K \to \mathbb{R}$, C^∞
- $\sigma : \mathbb{R} \to \mathbb{R}$, C^∞, NON-POLYNOMIAL OR BOUNDED

$\forall \varepsilon, \exists f : K \to \mathbb{R}$ (1-HIDDEN-LAYER NET.) ST. $\sup_{\bar{x}\in K} |f(\bar{x}) - g(\bar{x})| < \varepsilon$

$\left(f(\bar{x}) = \sum_{j=1}^h C_j \sigma(\bar{\omega}^{(j)}\cdot\bar{x} + b^{(j)})\right)$

- NO BOUND ON h SIZE ↑ TO AVOID EXP SIZE, DEEPER ARCHITECTURES ARE USED
- MLP HAVE $\text{VC}_{\dim_{\text{MLP}}} < \infty$ (HENCE AGN-PAC-LEARN)
- UAT SAYS ε_{app} CAN BE LOWERED ARBITRARILY ⇓ NETWORKS AS UNIVERSAL FITTING DEVICES

MLP ARCHITECTURE

COMPOSITION OF PERCEPTRONS. L LAYERS, WHERE EACH LAYER MAPS → $\mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$, $\bar{x}^{\ell-1} \mapsto \bar{x}^\ell$

$\bar{x} = \bar{x}^0$ IS FIXED AS INPUT, $K = |\mathcal{Y}|$ IS FINAL LAYER SIZE.

SOFTMAX (LAST LAYER)

$P_y(\bar{x}^L) = \frac{e^{x^L_y}}{\sum_{j=1}^{N_L} e^{x^L_j}}$

SOFTMAX IS ONLY USEFUL IN TRAINING → argmax IS SAME

NORMALISES, THEN USE CE LOSS → $\mathbb{P}_y$ IS LOWERED BY REDUCING x^L_y OR ALL THE OTHER $x^L_{\neq y}$

MATRIX NOTATION

$W\bar{x}^{\ell-1} + \bar{b} = \bar{z}^\ell$ ($W \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$) (PRE-ACTIVATIONS)

$\bar{x}^\ell = \begin{cases} \sigma(\bar{x}^L) & \ell < L \\ \bar{x}^L & \ell = L \end{cases}$

- DENSE LAYERS ARE THE MOST EXPENSIVE, BUT MOST GENERAL
- USED FOR THE FINAL LAYERS OF COMPLEX ARCHITECTURES.
- FEED-FORWARD ARCHITECTURE IS THE ACYCLIC GRAPH ONE

ACTIVATION FUNCTION

→ σ AFFINE ⇒ WHOLE NETWORK REDUCES TO AFFINE

→ SIGMOID (LIKE tanh) AS AN APPROXIMATION OF SIGN (AKA. STEP FUNCTION)

→ RECTIFIED LINEAR UNITS (ReLU) $\text{ReLU}(x) = \max(0, x)$ (SOLVES THE VANISHING GRADIENT PROBLEM) | GeLU AS A SMOOTH APPROXIMATION

SYMMETRIES

THERE ARE TRANFORMATIONS OF THE PARAMETERS THAT PRESERVE THE FUNCTION.
- DISCRETE: PERMUTING INDICES OF UNITS. $(\Pi_{\ell=1}^{L-1}(N_\ell)!$ eq. CLASSES)
- CONTINUOUS: STUFF LIKE $+c$ AT THE LAST LAYER, OR DEPENDING ON ACTIVATION: $\alpha > 0 \to \text{ReLU}(\alpha x) = \alpha\,\text{ReLU}(x)$ SO INVERSE SCALING CAN BE INCLUDED IN $W^\ell, \bar{b}^\ell$

HENCE MLP PARAMETRISATION IS REDUNDANT AND LOSS CANNOT BE STRICTLY CONVEX LOCALLY, DUE TO THE EQUIVALENT SYMMETRIES INDUCED BY SYMMETRIES. (BUT $\#\text{SYM} \ll \#\text{PARAMS}$)

PERFORMANCE
- GPU HARDWARE IS OPTIMISED FOR MATRIX MULTIPLICATION.
- YOU CAN ALSO COMPUTE MULTIPLE MULTIPLICATIONS AT ONCE BY EXTENDING / DUPLICATING THE MATRIX.
- BROADCASTING $\sigma(\cdot)$ IS ALSO EMBARASSINGLY PARALLEL.

GRADIENT DESCENT

ASSUMPTIONS → $\mathcal{J}$ CONTINUOUS, NBOUNDED → LOSS $\in C^2$ μ-ae

$f : \mathbb{R}^N \to \mathbb{R}$, FIND $\bar{x}^* = \text{argmin}_{\bar{x}} f(\bar{x})$

ITERATIVE ALGORITHM:
- SAMPLE $\bar{x}^{t=0}$
- $\bar{x}^{t+1} = \bar{x}^t + \delta\bar{x}^t$ UP TO CONVERGENCE
  - FIRST ORDER → USES ∇f (GRADIENT DESCENT)
  - SECOND ORDER → HESSIAN, (NEWTON METHOD)

GRADIENT DESCENT

$\delta\bar{x}^t = -\alpha_t \nabla f(\bar{x}^t) \,|\, \alpha_t > 0$

APPROXIMATE f IN A NEIGHBOURHOOD AS THE TANGENT, FOLLOW STEEPEST DIRECTION.

α_t SUFF. SMALL WILL DECREASE OBJECTIVE LOSS (GENERALLY DECREASES IN t)

$\forall x, \exists \varepsilon : \alpha < \varepsilon$, $f(\bar{x} - \alpha\nabla f(\bar{x})) \leq f(\bar{x})$

f L-LIPSCHITZ, $\alpha_t < \frac{1}{L}$ CONVERGES IN $O(\frac{1}{t})$. (ASSUMING ∃! GLOBAL MINIMIZER)

→ SUB-GRAD. CAN BE USED IF NOT DIFFERENTIABLE

ISSUES:
- OVERSHOOTING FOR $\alpha > L^{-1}$ (MAY CAUSE OSCILLATIONS)
- SLOW CONVERGENCE ALONG SOME DIRECTIONS DUE TO GENTLE SLOPE

NEWTON METHOD

$\nabla f(\bar{x}^t + \delta\bar{x}^t) \approx \nabla f(\bar{x}^t) + H_f(\bar{x}^t)\delta\bar{x}^t$

APPROXIMATE f NEAR $\bar{x}^t$ USING THE TANGENT PARABOLOID, SET $\bar{x}^{t+1}$ TO THE VERTEX. CONVERGES QUICKLY IN CONVEX SETTING.

COSTLY: $O(h^2)$ FOR H, $O(h^3)$ FOR MATRIX INVERSION → EVEN APPROXIMATIONS ARE $O(h^2)$ MEMORY (QUASI-NEWTON)

GD IN NON-CONVEX
- FINDS LOCAL MINIMUM, DEPENDS ON $\bar{x}^0$
- TYPICAL PROBLEM OF GRADIENT TENDING TO ZERO.

GD WITH MOMENTUM:

$\delta\bar{x}^t = -\alpha_t\nabla f(\bar{x}^t) + \beta(\delta\bar{x}^{t-1} - \alpha_t\nabla f(\bar{x}^t))$

$\beta \in [0.8, 0.9]$ TYPICALLY

DAMPS OSCILLATIONS BUT MAY OVERSHOOT, BY ADDING OSCILLATIONS AT END

NESTEROV MOMENTUM

$\delta\bar{x}^t = -\alpha_t\nabla f(\bar{x}^t + \beta\delta\bar{x}^{t-1}) + \beta\delta\bar{x}^{t-1}$

LOOK-AHEAD TERM COMPENSATES MOMENTUM, REMOVING OVERSHOOTING. EASY TO IMPLEMENT

$\beta \in [0.8, 0.9]$ TYPICALLY, ON CONVEX, $\varepsilon = O(\frac{1}{t^2})$

ADAM (ADAPTIVE PER-VARIABLE RATE)

ADAPTS EACH VARIABLE RATE BASED ON LANDSCAPE. FAST, BUT NOISY. DEPENDING ON DATA, COULD BE WORSE THAN NESTEROV.

$(\beta_1 = 0.99,\ \beta_2 = 0.999,\ \varepsilon = 10^{-8})$

$\nabla f(\bar{x}^t) \to \bar{g}^t, \bar{m}^t, \bar{v}^t$

3 AUXILIARY VECTORS + $\beta_1, \beta_2, \varepsilon$ HYPERPARAM

$\bar{m}^t, \bar{v}^t$ ARE ROLLING EXP AVERAGES OF GRADIENT, MOMENTUM

EXPONENTIAL ROLLING AVERAGE

$a^t = \frac{1-\beta}{1-\beta^t}z^t + \beta\frac{1-\beta^{t-1}}{1-\beta^t}a^{t-1}$

- $O(1)$ MEMORY
- WEIGHTED AVERAGE, AS $\beta \to 1$ THE AVERAGE BECOMES UNIFORM

$\frac{1-\beta^{t-1}}{1-\beta^t}$: DECAY OF TERMS; CONTAINS PREVIOUS TERMS; NO NEED TO STORE THEM

EMPIRICALLY, THE FIRST ITERATIONS HAVE THE LARGEST EFFECT → SUCCESSIVE ARE THE REFINEMENTS

STOCHASTIC GRADIENT DESCENT

SAMPLE THE GRADIENT FROM A SET, INSTEAD OF USING ALL THE DATA → INDUCES RANDOMNESS: TRADEOFF BETWEEN STABILITY AND SPEED

$\nabla L_S(\vartheta) \approx \frac{1}{|B|}\sum_{\mu\in B}\nabla L_\mu(\vartheta)$

MINI BATCHES ARE FED IN RANDOM ORDER AND SHUFFLED.

SGD IS JUST A NOISY GRADIENT ESTIMATE SO IT CAN BE COMBINED WITH ALL OTHER METHODS

CONVERGENCE IS $O(\frac{1}{\sqrt{t}})$, BUT NOISE HELPS FIND "STABLE" & LOCAL MINIMA, AVOIDS

SGD HYPERPARAMETERS

CROSS-VALIDATION CAN BE USED TO FIND THE BEST HYPER-PARAMETERS:
1. SPLIT DATA IN k (EACH IS A "FOLD")
2. TRAIN k TIMES, LEAVING OUT A FOLD, WHICH IS USED FOR VALIDATION
3. AVERAGE VAL. ERR. OVER k → CROSS-VAL. ERR.

EXPLORE THE RANGE OF HYPERPARAM (EX. GRID SEARCH)

k SHOULD BALANCE SPEED AND STABILITY. USING THE WHOLE TEST DATA IS RISKY

INITIALISATION ← RELEVANT TO NON-CONVEX PROBLEMS

IN MLP SYMMETRIES WILL MAKE THE GRADIENT THE SAME ⇒ $\nabla f(\bar{x}) = h.\begin{pmatrix}1\\1\end{pmatrix}$

IF YOU SAMPLE $\omega \sim N(0,1)$ OR $U(-1,1)$, THE VARIANCE OF THE VARIANCE OF THE LAST LAYER'S ACTIVATION DIVERGES EXP IN #LAYERS

VERY PEAKED OR VERY FLAT DISTRIBUTIONS ⇒ ONE WANTS $v^\ell \simeq \frac{2}{N_{\ell-1}}$

GLOROT: UNIFORM THAT ACCOUNTS FOR INPUT/OUTPUT OF LAYER $U\left(\pm\frac{\sqrt{6}}{\sqrt{N_\ell + N_{\ell-1}}}\right)$

KAIMING INITIALISATION $W^\ell_{ij} \sim N(0, \frac{2}{N_{\ell-1}})$

$\sqrt{2}$ DEPENDS ON σ

SCALE $\sim \sqrt{\frac{2}{N_{\ell-1}}}$

STOPPING CRITERIA
- SGD → IF $\|\nabla f(\bar{x}^t)\| < \varepsilon$ FOR ε CHOSEN BEFORE
- STOP WHEN LOSS GETS SMALL ENOUGH / CLOSE TO ZERO
- ONE TENDS TO SAVE CHECKPOINTS

LEARNING RATE SCHEDULE

YOU KNOW IT SHOULD DECREASE OVER TIME
- **STEP DECAY:** $\alpha \to \frac{\alpha}{m} \to \frac{\alpha}{m^2} \cdots \to \frac{\alpha}{m^5}$ (TRANSITIONS FROM EXPLORATION TO EXPLOITATION)
- **COSINE DECAY:** $\frac{\alpha}{2}\left(1 + \cos\left(\pi\frac{t}{t_{\max}}\right)\right)$ CONTINUOUS DECREASES (SPEND TIME ON HIGH AND SLOWS DOWN TOWARDS END)
- **KNEE SCHEDULE:** $\frac{t_{\max}}{2}$ → HIGH α VALUES; REST OF THE α DECREASES LINEARLY
- **BILEF WARMUP:** INCREASE α FROM 0 TO α BEFORE STARTING SOME SCHEDULER

MINI-BATCH SIZE (SMALLER BATCH ⇒ FASTER, NOISIER)

$W\cdot\bar{x}$ IS COSTLIEST OPERATION → MADE MORE EFFICIENT BY STACKING THE $\bar{x}$ INTO A SINGLE, LARGER MATRIX → SO WITH GPU LARGER BATCH SIZES ARE PREFERRED

ALSO TOO LARGE m_B IS TOO STABLE IN SGD

Automatic Differentiation

Differential programming: write a program with free parameters, differentiate the output wrt parameters. (Python → Autograd, JAX)

Symbolic diff — computer algebra system (CAS) → algorithm that takes an expression and returns the derivative (substitution rules)
- solution may be inefficient → caching/heuristics needed
- cannot handle arbitrary logic (if/else, loops)

Forward diff

1. Static single assignment (SSA)
$\cos(x_1^2 + x_2^3) = x_3$
$w_1 = x_1$
$w_2 = x_2$
$w_3 = w_1^2$
$w_4 = w_2^3$
$w_5 = w_3 + w_4$
$w_6 = \cos(w_5)$
2. Construct the computational graph (DAG)
3. Differentiate each line wrt 2 (unspecified)
4. "Seed" aka set $\frac{\partial x_i}{\partial x_j} = \delta_{ij}$

The differentiated graph has the same topology as the function → you also don't need to store previous computations

The algorithm grows in #vars, remaining efficient in large dimensional outputs.

Reverse diff

Compute the adjoint of each w_i var: $\bar{w}_i = \frac{\partial f}{\partial w_i}$ → trivial at the last step of computation ($\bar{\delta}_{ij}$)

Go backwards, up to gradient

2x passes are needed:
1. Fw: compute w_i values
2. Bw: compute the gradient

$O(m)$ since you need to compute $f(x)$ m times

↑ you can also combine reverse/forward diff depending on what is more tractable

Dual numbers

$\{a + \varepsilon b\} \cong \mathbb{R}^2$ (finite part, infinitesimal part $\varepsilon^2 = 0$)

The Taylor series can be used to extend function domains to dual numbers

Use overloading in programming to compute derivative in one step

$P(x)$ polynomial has $P(x + \dot{x}\varepsilon) = P(x) + \dot{x}P'(x)\varepsilon$

$f(x + \dot{x}\varepsilon) = f(x) + \dot{x}f'(x)\varepsilon$

Finite difference

For f black box. Approximate $\frac{\partial f}{\partial x_i}$ using $f(x+\varepsilon)$

(May cause float errors, since ε needs to be $<<1$)

Control flow

Forward diff works on general code (even with if blocks)

Similarly, dual numbers work.

(Reverse diff) tapes

"Tapes" record all the intermediate values and the computational graph.
- (Fw → record data)
- (Bw → use data)

Computing a operation saves the value but updates the tape with variables used to compute the result (this also extends to control flow)

Tapes work as long as every step is documented in a new object

(Reverse diff) source-to-source

Works on code itself: takes the function definition and constructs the code of the derivative, afterwards

Backpropagation ($\vartheta \in \mathbb{R}^n$, $n = \sum_{l=1}^{L}(N_{l-1}+1)N_l$)

Since $f : \mathbb{R}^n \to \mathbb{R}^2$, it makes sense to use reverse diff

(By assumption, $f^l = \sigma(z)$ $l \le L$, $f^L = z$)

$\bar{x}^0 = \bar{x}^{in} \to$ input is 0 activation

$\bar{z}^l = W^l \bar{x}^{l-1} + b^l$

$\bar{x}^l = f^l(\bar{z}^l) \to$ activation

$L_\mu = -\sum_y q_y^* \left(\bar{x}_y^L - \log \sum_{y'} e^{\bar{x}_{y'}^L}\right)$

Reverse diff → $\bar{x}_i^L := -(q^* - \bar{p})^T$ (error signal)

$\bar{z}^l = \frac{\partial l}{\partial \bar{x}^l} \cdot \frac{\partial \bar{x}^l}{\partial \bar{z}^l} = \frac{\partial l}{\partial \bar{x}^l} \cdot \frac{df^l}{dz}(\bar{z}^l)$

$\frac{\partial l}{\partial W^l} = \bar{x}^{l-1} \frac{\partial l}{\partial \bar{z}^l}$

$\frac{\partial l}{\partial b^l} = \frac{\partial l}{\partial \bar{z}^l}$

$\frac{\partial l}{\partial \bar{x}^{l-1}} = \frac{\partial l}{\partial \bar{z}^l} W^l$

Vanishing gradients

If activation is a saturating function, the error is not propagated to the first layers (slows learning) → solution to use ReLU

Computation

Computing gradients is also grouped in matrices for efficiency

Regularization

Goal

One wants to minimize generalization error. Training error is an intermediate step.

Just lowering train E. leads to overfitting

The loss landscape has many local minima of similar value but different gen. error.

Training regime "biases" the model towards some minima (the best training can find good minima reliably)

$S = \{(\bar{x}^n, y^n)\}_n \sim \mathcal{D}(\bar{x}, y)$

A stocastic algorithm → depends on hyper-params

t training time → output of alg: $\vartheta_A(S,t)$

Generalization: $\mathrm{err}_{\mathcal{D}}(\vartheta_A(S,t))$

Error → $\hat{E}(t) = \mathbb{E}[\mathrm{err}_{\mathcal{D}}(\vartheta_A(S,t))]$

$\mathbb{P}(\varepsilon, t) = \mathbb{P}(\mathrm{err}_{\mathcal{D}}(\vartheta_A(S,t)) < \varepsilon)$

- Ideally, with limit $\hat{E}(t)$ $t \to \infty$
- Time ⇒ given matters & budget: with $\hat{E}(t)$ ok for ε acceptable, $\max$ limit $\mathbb{P}(\varepsilon, t)$ — or —

→ Generalisation is also in robustness to perturbation: define $\bar{\delta}$ and use it for validation

Other ways like GANs

Train to make best use of budget

Overfitting

Early stopping

Train as long as val-acc is improving
- take a rolling avg for fluctuations
- in overparam. NN error with just plateau

Prior

Choice of prior is not obvious. In CE loss, add $-\frac{1}{m}\log \mathbb{P}(\vartheta)$

L_2 reg

Gaussian iid prior $\vartheta \sim \mathcal{N}(0, \varepsilon)$ will produce flatter distributions, hence having a prior makes the model more cautious

Weight decay

Using l_2 reg, in GD, the added contribution is $-\lambda\bar{x}$ aka. a push to 0. (the name comes from $\bar{x} \to 0$ without $\bar{\nabla}f$ term)

AdamW

With l_2 reg, Adam performs better using the same rate

Gradient Cupping

If loss is highly irregular, very high gradients can make the parameters jump.

$\bar{\nabla}f \to$ clamp each $\bar{\nabla}f_i$ between $\pm \max$

$\bar{\nabla}f \mapsto \bar{\nabla}f \cdot \frac{\max}{\|\bar{\nabla}f\|}$ if $\|\bar{\nabla}f\| > \max$ (rescale to max)

Label Smoothing

"Relax" the one-hot label vectors, so they don't produce just peaked predictions.

$q'_y = (1-\varepsilon)\delta_{y,y^n} + \frac{\varepsilon}{131}$

Logit

Add a term $+2118^4\|\vartheta\|^2$ so last layer is smaller

Won't learn the exact soft labelling structure

Batch Norm (CNN>>)

Variance of activation propagates exponentially wrt # layers (a similar thing happens in the backpropagation of error)

Normalise μ, σ^2 per-batch, at a layer ($z = \frac{x-\mu}{\sigma}$) → use in validation the μ, σ^2 over the whole dataset

$\gamma z + \beta$

Epochs take longer but network converges faster

(avoids the need for the NN to normalise features)

As an affine transform. It could be applied in the previous layer directly

Layer Norm

If a layer is reused, normalization happens by pattern. (perceptron is very similar.) ← in RNN

Covariate shift

Variance increases over layers, so training is slower because it is hard to anticipate behavior

↓

Normalization acts basically like "guiding" input values using a funnel

Dropout

Ensembling: train more models and pick/ masonry vote

Mistakes should be few/uncorr. → quite costly

In feedforward, add a dropout layer $\mathbb{R}^m \to \mathbb{R}^m$ which in training is $\frac{x_i}{1-p}$ with $\mathbb{P} = 1-p$, otherwise 0. in val, it becomes E. usually $p \le 0.5$

- encourages multiple internal encodings
- simulates averaging

Wide Minima

Correlate to better generalization (overparam networks)

↓

No consensus yet on best way to bias algorithms

Data Augmentation

Use assumptions on training data to expand train set

Assume smoothness of $\mathcal{D}$, so if you perturb x, $\mathcal{D}(x+\varepsilon) \approx \mathcal{D}(x)$ (then you sample ε and produce $x+\varepsilon$)

- Domain-specific: perturb without affecting class (scale, color, rotate image)
- Generic: use the other points in dataset

Mixup: $\lambda \sim Beta(\alpha,\alpha)$ then construct a convex combination between $2x_i, x_j$. (distr. is concentrated on ends, so pattern remains famous) ← $\alpha \in [0.1, 0.4]$

Experience replay (show past examples to avoid catastrophic forgetting (breaking past memories) ↑

Reinforcement Learning

Settings

- Value function that evaluates the current state of the board → $V(s) = \mathbb{P}(\text{win})$, pick s' next move st. $V(s)$ highest
- Function that, given the board, produces $\mathbb{P}$ over moves → aka. Policy: $\pi(a|s)$ (action | state)

Challenges

- $S \to$ encode as an image with a range of values
- $A \to$ one hot on vector/other

(+ use record of tournaments → to emulate for training — train to emulate win. moves)

Basics

Agent A subjectively interacting with the environment E through actions. At each t:
1. A has state $s_t \in S$
2. From the state, A performs an action $a_t \in A$ ⇒ instantaneous reward (negative too)
3. $s_t \mapsto s_{t+1}$

Evolution of system

$(S \times A \times \mathbb{R} \times S)$ (state before, state after)

Stochastic variant

- $\pi(a|s) = \mathbb{P}(a_t = a | s_t = s)$
- r_t reward can be stochastic, so use $\mathbb{E}[r_{t+1} | s_t = s, a_t = a]$
- Random state transition $p_0 \sim p_0$, $P(s'|s,a)$

Sources of randomness: (no knowledge, unpredictable agents, ...)

Possible challenges

- Sparse rewards, so greedy strategy is not a good idea
- The current state may need to include past history (choice of how much)
- Some actions may be forbidden in the current state

Loss

Maximise cumulative reward:

$G_t = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} = r_{t+1} + \gamma G_{t+1}$ (current loss is given by the future)

$\max \mathbb{E}[G_0]$ (aka future reward from the start) — there's also dealing with very large action spaces

Expected future reward, given $s, a | \pi$

$Q_\pi(s,a) := \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$

Dependency on time can be dropped (Markov process) → more simply

$V_\pi(s) := \mathbb{E}_\pi[G_t | s_t = s]$ (value of state) $= \sum_a \pi(a|s) Q_\pi(s,a)$

Policy gradient

Optimise $\pi(a|s)$ as $\pi^* = \arg\max_\pi \mathbb{E}[G_0]$

or if $s_0 \sim P_0(s)$, $\pi^* = \arg\max_\pi V_\pi(G_0)$

Pol. grad. thm

$\nabla V_{\pi_\vartheta}(s_0) = \sum_s \mu_{\pi_\vartheta}(s) \sum_a \bar{\nabla}\pi(a|s,\vartheta) Q_{\pi_\vartheta}(s,a)$

$= \mathbb{E}[\bar{\nabla} \log \pi(a|s,\vartheta) Q_{\pi_\vartheta}(s,a)]$

No D_ϑ, D_Q are present, so it's implementable

Learning depends a lot on sampling, which is generally expensive (Monte Carlo)

Q-learning

If the reward is Markovian, the π^* is deterministic

$Q^*(s,a) = \max_\pi Q_\pi(s,a)$

$\pi^*(a|s) = \begin{cases} a & a = \arg\max_{a'} Q^* \\ 0 & \text{else} \end{cases}$

$\pi(s) = \arg\max_a Q^*(s,a)$ (leads to the most payoff)

Q-learn. challenges

$r(s,a)$, $P(s'|s,a)$ are unknown, but can be sampled

$\mathcal{L}(\vartheta) = \left((Q_\vartheta(s_t, a_t) - r_{t+1} - \gamma \max_{a'} Q_\vartheta(s_{t+1}, a'))\right)^2$

- When sampling, it's hard to get good examples (exploration vs exploitation)

Experience replay (show past examples to avoid catastrophic forgetting (breaking past memories) ↑

Convolutional Neural Networks

MLP are limited by permutation-invariant structure. (relational information could be exploited) → see in images, pixels are correlated

A MLP would have to learn for a long time to replicate all symmetries

Instead of augmenting the data in a transl. invariant way, use a suitable operation.

Convolution filter

- Kernel → the ideal pattern (W)
- Feature map → the result of the dot product between kernel and input, by applying the dot $\bar{\vartheta}$ at every neighbourhood of the image.

(ex. vertical edges) $W = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & -1 \end{bmatrix}$

Generally, W is square.

Img: $w \times h$

Feature: $(w-2) \times (h-2)$

Multiple channels

One convolution per layer (eg. color), but the kernel does not convolve across layers.

↑

Later layers can convolve again. The kernel itself is learnable, together with weights and biases (flattening + MLP at last layers)

Receptive field

Each pixel in next layer has had access to only h^2 of the previous. (this neighbourhood) is called R. field

↓

The next layer will have involved a $(h_1 + h' - 1)^2$ square

↓

If h^L is size of W, the L^{th} layer has side: $\min\left(1 + \sum_{l=1}^{L}(h^l - 1), w\right)$ ↑ increases up to covering whole input

Considerations

In order for the final layer to have the largest receptive field, you need
A) large R. field
B) deep network
C) both

d-dim case

Generalisation is easy
- data in grid
- video / time dim
- 3d video

each dim may have a different kernel size

Padding

Applying a kernel produces a reduced representation: $w_a - k_a + 1$ is the next layer's side.

A "border" can be added to keep the next layer of the same size (enlarge by $k_a - 1$, placing per side $\left\lfloor \frac{k_a-1}{2} \right\rfloor$, $k_a - 1 - \left\lfloor \frac{k_a-1}{2} \right\rfloor$)

Pad techniques

- set as 0
- mirror image
- domain-specific augmentation

Striding / downsampling

Reduce complexity by making the kernel "jump", usually applied uniformly in all directions. $s \ge 1$ (increases R. field)

Pooling

Like kernel, but instead of a dot P. it's a node general function that reduces dim. (max, $\mathbb{E}$) (usually has h-stride, effectively partitions the layer. usually $s \ge 1$)

Features of Pool.

- Bigger R. field
- You can add node channels with the saved compute cost
- Helps with translation invar.
- Typically, many pooling layers are used

Equivariance wrt translation

If data $x \in \mathcal{X}$ input space with some symmetry. ↑ A symmetry is $T \in \mathcal{T}$ → transformation $f : \mathcal{X} \to \mathcal{X}$

If $\mathcal{X}, \bar{\mathcal{X}}$ have the same symmetries → f equivar. if $\forall T \in \mathcal{T}$ $f \circ T = T \circ f$

Equivar is closed w composition → equivar is good in cann since it ensures transl. invariance

CNN as sparse linear

Inductive bias (you know images are T. invariant)

As it is locally a dot P., can is a sparse matrix.

% nonzero entries $= \frac{k_1 \cdot k_2}{w \cdot h}$

Matrix shares weights, which further reduces memory requirements

Deeper Layers

- Deeper → better, up to ~19 (see VGG19), but train err. is eventually higher (more irregular loss)
- Skip connection → a deeper layer also takes an earlier layer as input. (requires that layer size is preserved)
- Residual block (ResNet) → block of layers encompassed by a skip connection → network becomes shallower, so trainable

Effects of Res

$x \mapsto x + \mathcal{S}$ (residual)

Many small distortions are applied.

If layer dims do not match, a linear transformation W can also be used to upscale

Squeeze and Excitation

Too deep ResNet → more efficient learning and use of params

$w_c = SE(x_c)$ values are computed per channel, which strengthen features.

May learn to suppress features ("soft gating")

Language Models

Natural Language Processing

- classification → sentiment analysis
- translation
- summarization
- generation

Problems with tokens

- You need a lower # characters
- Use placeholders for "unknown" word
- Agglutinating languages

Byte pair encoding (BPE)

Start with a token dict (eg. 256) and grow it with tokens that are always consecutive. Final size ~ 10^4 but tokens will be meaningful

Embeddings

Each word has a number, but you can't just use the numbers. ↓

Each char/word → one-hot encoding

Use $W \times$ one-hot to tokenize the sequence into a very sparse matrix. Project this W^E to lower dimension: $X = W^E T$

Then X preserves some logic in the vs: woman + crown = queen (but ambiguity based)

Vectors to text

1. Use W^E transposed to enlarge back
2. Apply softmax to vector and sample the next word

(W^E is learnt in 2v during training)

(if #tokens changes, use two matrices (eg. translation))

Recurrent (seq2seq)

$\bar{z}, \bar{h}^t = f(\bar{x}, \bar{h})$ ↑ arbitrarily deep NN

Usually you need later words to do the task.

(SOTA pre-transformers)

Feed the input sequence one vector at a time.

$\bar{h}^{(1)} = \bar{0}$, then let $\bar{h}^t$ evolve (stores the internal state)

$\bar{x}$ here is the one-hot word

Encoder-decoder

$\vartheta = (\vartheta^E, \vartheta^D)$ params

Two require netw. are used.

- Encoder: ignore $\bar{z}^t$ output → constructs $\bar{c}$, describing the whole sequence
- Decoder: $\bar{h}^{t+1} = \bar{c}^{(t+1)}$ → "context" is used to recreate the new sequence. $\bar{z}^{(t)} = \bar{0}$ (use a "stop" token for end-of-sequence)

Backprop

Size of compute. Graph is not known beforehand. Store calculations in time (costly)

Difficulties

Expensive to run and train. Sequential means hard to parallelise

Multi-headed attention

In attention, most steps are linear → there are multiple aspects to the task

Multiple kinds of attention, in parallel (each called "head") → concat to shrink vector at the end

params is now multiplied by #heads. $d_c = d$, $d_v = d/h$

$O(d^2)$ for tot params → $d_k = d/h$

Since order matters in language, one will modify attention or just encode position as a feature

Long-range interactions

Self-att. module → each output vector is a function of all input vectors (regardless of #vectors)

Problems: $O(n^2)$ in context size, cross-att. invariant to permutations of C cols

Self-att. equivar to perm. X cols

Permuting keys does not affect aggregate results → reorder input will reorder outputs

Queries, keys, values

d, d_c large ⇒ J, P large, costly in memory → split into lower-rank matrices

$J = W^K \bar{W}^Q$ (key, query)

Memory from dd_c to $(d+d_c)d_k$

$d_k \ll d, d_c$ yield speedup gains (same holds for P project.)

$\bar{x}' = W^V \bar{x}$ (value)

$\bar{w} = \frac{K^T}{\sqrt{d_k}} \bar{q}$ (scaling helps in practice)

Used to scan the context ↓ matching score

$\bar{z} = A\bar{c}c(\bar{x}, \bar{c})$

$\bar{w} = C^T J$ (m) $(h \cdot d_c)$ $(d_c \cdot d)$ (d)

$\bar{\alpha} = \mathrm{softmax}(\bar{w})$

$\bar{u} = C \bar{\alpha}$ $(d_c \cdot w)$ (w)

$\bar{z} = P \bar{u}$ $(d \cdot h d_c)$ (d_c)

$\bar{x}' = \bar{x} + \bar{z}$

d, d_c, m hyperparams

Add $\bar{z}$ to activations

Positional Encoding

If the same vector appears twice, att. will not distinguish it. Add a feature.
- one more dim in $\bar{x}^0$, containing index
- add a fixed W^P to embedded inputs. (learnt as well)

Requires n_{max} for length of sequence, but last cols receive less training

Why self-attention?

If recursive seq2seq scheme is replaced,
- Recursive just looks at previous inputs, self-att looks at the whole seq.
- Self-attention has no natural encoding of position
- Self-attention is parallelizable because it avoids recursion

In recursive, the J^{th} output must encode the whole sequence up to j ↓ adding attention, each token gets a representation, but feature extr. is needed. Context produces different embeddings even for the same token

Training challenges

Even with self-att, decoding is hard to parallelize because output length is not known before. ↑

At train, length is known beforehand.

Train to predict next word, as an autoregressor

Introduce a dummy "start"

Parallelizes training.

Testing uses encoder for context, sampling the continuation until "stop". → cache the computations

Attention → **Pros**

During decoding, the model should have access to a summary of the whole sequence ↑

$\bar{c}^{(n+1)} = \bar{h}^{(n)}$ is also a summary, but its size is fixed in encoding

An attention weight is associated to all context vectors and used to combine them (note, all contexts are used ⇒ full info.)

- Fixed $\bar{c}$ vector size, but importance is dynamic.

Simple attention

You need $w \times$ attention weights, one per context vector → $\bar{a} = (a_j)_{j \in [n]}$

$a_j = \phi(\bar{x}, \bar{z}^j)$ can be used to produce a weighted average (positive, then ⇒ $\alpha_j \in (0,1)$ divide by sum)

↓

Projection matrix to expand the vector to a comparable size

ϕ can always be composed with a softmax to get $\mathbb{P}$

ϕ could be $\bar{z}^{(j)} \cdot \bar{x}$, generalising dot P. for differently sized vectors (learn the projection matrix)

Add $\bar{z}$ to Activations

$$\text{Attention} = \text{Softmax}\left(\frac{KQ}{\sqrt{d_k}}\right)V$$

(optimised for info. representation; optimised to express the next word)

Transformer Blocks

Heuristics: (use a cheap σ, move layer norm before att/MLP)

Encoder block: sequence of attention modules, followed by shallow MLP (1-2 $\oplus$ 2 usually), finally layer norm. MLP has a large hidden layer but preserves in-out dim

Decoder block: self and cross attention + layer norm, MLP_a

Masked self-attention

Masking $\oplus$ shift right of tokens → add to self-att a mask matrix: $M_{ij} = -\infty \mathbb{I}(i \le j)$

Causal masking only uses past tokens in model

Self-supervised transformers

→ BERT: encoder-only (typical task to reconstruct a fragmented text) ↑ use the block as text preprocessing

→ GPT: decoder-only ↑ no cross-attention, can complete tasks as long as you provide the beginning

1

- CLASSIFICATION →
  - ACCURACY: CORRECT / ALL PREDICT } SENSITIVE TO IMBALANCED DATASET
  - PRECISION: TP / (TP+FP) } MINIMISE FALSE POSITIVES → IF THEY ARE COSTLY
  - RECALL: TP / (TP+FN) } MINIMISE FALSE NEGATIVES
  - F1-SCORE → HARMONIC MEAN BETWEEN PRECISION AND RECALL } MINIMISES THE DIFFERENCE BETWEEN THEM
  - FALSE POSITIVE RATE → MINIMISE FALSE POSITIVES
- REGRESSION → MEAN ABS ERR (MAE), MEAN SQUARED ERROR (MSE)
- RECEIVER OPERATING CHARACTERISTIC (ROC) CURVE → TRADEOFF BETWEEN SENSITIVITY AND SPECIFICITY (TPR, FPR)

2

- SCIKIT LEARN → NO GPU, NO SCALABILITY
- TENSORFLOW → STEEP LEARNING CURVE
  - (KERAS) → HIGH LEVEL API ON TENSORFLOW
- JAX → JIT COMPILATION, FUNCTIONAL, AUTOGRAD } NUMERICAL COMPUTING
  - (FLAX) → NN LIBRARY
  - (NNX) → SCALABLE INTEGRATED, MODULAR NN EXTENSIONS
- PYTORCH → SIMPLE, SLOWER IN PROD, DYNAMIC COMPUTE, PYTHONIC SYNTAX

TENSOR → LIKE NUMPY NDARRAYS } TORCH.FROM_NUMPY() / T.NUMPY()
- ↳ REQUIRES_GRAD FLAG DECIDES TO STORE THE TAPE FOR GRADIENT

GPU COMPATIBILITY:
- T.TO(...)
- TORCH.CUDA.IS_AVAILABLE()

"OPTIMISER" OBJECT (SGD, ADAM, ...) ⊕ LOSS FUNCTION

MODEL CLASS (INHERITS FROM MODULE)
- __INIT__(SELF)
- FORWARD(SELF, X)

3

NODES — COMBINE INTO HPC } TASKS ARE DIVIDED AND RAN IN PARALLEL
- FASTER/DEEPER RESULTS, BUT EXPENSIVE, MORE ENERGY NEEDS

HPC CLUSTER (CENTRALISED) VS GRID COMPUTE (VOLUNTEERS) VS CLOUD COMPUTING (RENTED, SCALING UP/DOWN)

CPU (SEQUENTIAL TASKS, FEW CORES → TRANSISTORS PREDICT BRANCHES) VS GPU (MASSIVELY PARALLEL → DATA PIPELINES) } GPGPU AS A COMBINATION:
- SEQUENTIAL PARTS IN CPU, PARALLEL IN GPU
- BUT CODE CANNOT BE EXCHANGED

SIMD VS MIMD — SINGLE/MULTIPLE INSTRUCTIONS, DATA
- SIMD: IDEAL FOR PARALLEL TASK ON SIMILAR DATA → LOWER POWER COST
- MIMD: INDEP. PROCESSORS (HARD TO DESIGN, COORDINATION OVERHEAD)

CUDA (NVIDIA GENERAL PURPOSE)

CUDA COMPILER DIVIDES C++ INTO CPU, GPU CODE, EACH COMPILED DIFFERENTLY:
- C++ FUNCTION ("KERNEL") → COMPILED AND REPLICATED IN THREADS } ORGANIZED IN 3D "BLOCKS", CONTAINED IN 3D "GRID"
- A KERNEL IS ASSIGNED TO A BLOCK OF THREADS → EACH THREAD MAY RUN LOGIC
- BLOCKS DO NOT SYNC/COMMUNICATE BETWEEN } CPU MAY ASSIST

4

PYTORCH LIGHTNING EXTENDS GPU CODE TO MULTIPLE

6

TENSORBOARD INTERACTIVE VISUALISATION TOOLKIT (TRACK METRICS OVER THE RUNS) → JUPYTER, GOOGLE COLLAB
- SCALARS
- IMAGES → EG. DISTRIBUTION OF WEIGHTS AND BIASES
- HISTOGRAMS
- CONFUSION MATRICES

} LOG DURING FORWARD PASS, YOU CAN CODE CUSTOM LOGGERS IN PYTORCH

7

COMPUTER VISION
- HARD → CATEGORIES
- BACKGROUND CLUTTER
- ILLUMINATION
- DEFORMATION/OCCLUSION

IMAGENET (2009) → DEEP CNN WITH GPU (19x LAYERS)
- CNN WERE VERY POPULAR UP TO 2020

- TRANSFER LEARNING → ADD A BLOCK AT THE END AND ONLY TRAIN THAT
- ≠
- FINE TUNING → TRAIN FURTHER A PRETRAINED MODEL ON A NEW DATASET

8

OBJECT DETECTION
- CLASSIFICATION
- DETECTION → IF OBJECT IS IN IMAGE
- LOCALISATION → WHERE IS THE OBJECT } CNN TRAINED TO PRODUCE THE VERTICES OF BOUNDING BOX
- RECOGNITION
- SEGMENTATION } IMAGE ANALYSIS

TWO-STAGE DETECTOR → BACKBONE CNN → EXTRACTION OF PROPOSALS (REGION PROPOSAL) → CLASSIFICATION AND (MLP CLASSIFIER) LOCALIZATION (MLP REGRESSOR)

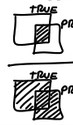
OR

SLIDING WINDOW → SUSCEPTIBLE TO SIZE, COSTLY

OR

REGIONAL BASED NETWORK → MANY BOXES ARE CHECKED, CLASSIFIED } SEARCH ALG. TO FIND BEST BOX (COSTLY)

SELECTIVE SEARCH → BASED ON SEGMENTATION, VARY ASPECT RATIO, SCALE (MERGE NEAR AREAS)



IOU METRIC → mAP (MEAN AVG PRECISION) → RANK PREDICTIONS IN DECREASING CONFIDENCE. COMPUTE PRECISION AND RECALL. AREA UNDER THE CURVE IS THE AP

REGION CNN (EXPENSIVE: CLASSIFY ALL PROPOSALS)

IMAGE → REGION PROPOSAL → NORMALISE IMG AND CLASSIFY IT

ONE-STAGE

NO REGION PROPOSALS → USE PRE-DEF REGIONS, CLASSIFY

TAKE ONLY THE MAX CONFIDENCE BOX → "NON-MAXIMUM SUPPRESSION" (TAKE MAX, DISCARD OVERLAPPING BOXES)

YOLO (2016)

BOXES + LABELS IN ONE EVAL (NON-MAX SUPPRESSION) } SINGLE-SHOT MULTIBOX DETECTOR
- MULTIPLE FEATURE MAPS
- DEFAULT BOXES, IP(CLASS) COMPUTED

FAST BUT SLIGHTLY LESS ACCURATE

10

TEXT (OR ANY UNSTRUCT DATA) → NLP → INFORMATION
- EMOTION
- TONE
- TOPIC

DIFFICULTIES:
- AMBIGUITY
- GAPS/NOISE
- STRONG ASSUMPTION OF COMMON KNOWLEDGE

BAG OF WORDS — DO ANALYSIS OF FREQUENCY-WORD PAIRS IN A TEXT } USE A VOCABULARY

PREPROCESS
- PUNCTUATION
- TYPOS
- TEXT CASING

→ LEMMATISATION → TURNS WORDS INTO THEIR ROOT CASE
→ STEMMING → SAME, BUT JUST CUTS THE END

PART OF SPECH (AKA. LOGICAL TREE, ADDED AS TAG)

TERM FREQUENCY → % APPEARENCE OF A WORD
×
INVERSE DOC. FREQUENCY → $\log\left(\frac{\#\text{DOCS}}{\#\text{DOCS W. WORD}}\right)$
‖
TF-IDF (FILTERS OUT COMMON TERMS)

DISCRETE WORD REPRESENT ≠ WORD MEANING

⇓

DISTRIBUTIONAL HYPOTHESIS (VECTORS ARE MAPPED TO LOWER, DENSE DIM THAT CAPTURE MEANING) → COSINE SIMILARITY

DOCUMENT EMBEDDING: WHOLE TEXT TURNS TO ONE VECTOR
- (COMPARE SIMILARITY)

11

CONTINUOUS BOW (MOVING MASK PREDICTS THE NEXT WORD THROUGH A DISTRIBUTION) → RELIES ON HUGE CORPUS, AND HIDDEN STATES WITH LOWER DIM

↕ REVERSE

SKIP-GRAM (PREDICT CONTEXT FROM TARGET) } TRAINED FROM A SLIDING WINDOW

CBOW TAKES PREVIOUS AND NEXT, PRODUCES MIDDLE

SKIP GRAM TAKES LAST WORD, PRODUCES THE PREVIOUS (WORKS BEST ON RARE WORDS)

RNN (FOR SEQUENCES)
- ATTENTION ASSIGNS DIFFERENT WEIGHTS TO WORDS, TO RETAIN RELEVANT INFORMATION
- PARALLEL, NO SEQUENTIAL BOTTLENECK

ENCODER ONLY (BERT)

DECODER ONLY (GPT)

BOTH (T5) } TASKS LIKE TRANSLATION

13

BERT (BIDIRECT. ENCODER) → VARIANTS FOR MORE LANGUAGES, BIGGER CORPUS

MASKED LANGUAGE MODELLING } TRAINED TO RECONSTRUCT TEXT

NEXT SENTENCE PREDICTION } CHECKS IF PHRASES ARE CONSECUTIVE

KNOWLEDGE DISTILLATION (TRAIN SMALLER MODEL FROM LARGER) → LOSS AS CVX SUM OF $CE_{LOSS} + KL_{DIV}$
- INTERNAL STRUCTURE MAY ALSO MATCH IN SMALLER MODEL

VISUAL TRANSFORMERS

BIGGER DATASET, BUT CAN FIND LONG-INTERACTIONS