Lectures

1 Compositional Models

1.1 End-to-End Learning

General purpose Machine Learning Desiderata → ***situation in DL (aka End-to-End Learning Paradigm):***

1. **Flexibility**: diversity of functions (few prior assumptions). → *modularity, compositionally, toolbox principle.*
2. **Adaptivity**: class of parametrized functions, generic learning. → *Restricted families of nonlinear functions, easy to define, good statistical efficiency, non-convex optimization.*
3. **Architecture**: modeling power vs. complexity trade-off (suitable design space/metaphors). → *In DL:*

- Layers of representation (width, depth, type): best practise, design patterns, informed exploration.
- Re-use of representations: multi-task, pre-training, AI etc.
- Input representation goal: less domain knowledge/feature craft, raw features (informative, but non necessarily explicit).

1.2 Compositional Models

Given function (implicitly): $F^* : \mathbb{R}^n \to \mathbb{R}^m$, via noisy samples $(\mathbf{x}_i \mapsto \mathbf{y}_i)$.
Learning: approximate F^* within class of functions $F : \mathbb{R}^n(\times\mathbb{R}^d) \to \mathbb{R}^m, \mathcal{F} \coloneqq F(\cdot,\theta)$. (weights θ, d-dimensional family, model selection: choose suitable $\mathcal{F}$, model fitting: find best $F \in \mathcal{F}$ such that $F \approx F^*$

1.3 Elements of Computation

Linear function: Simplest non-trivial functions. *Weighted summing* of inputs. A function $f : \mathbb{R}^n \to \mathbb{R}$ is a linear function if the following holds:
$f(\mathbf{x}+\mathbf{x}') = f(\mathbf{x}) + f(\mathbf{x}')$, $(\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^n)$ and $f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})$, $(\forall \alpha \in \mathbb{R})$
Proposition: f linear $\Leftrightarrow f(\mathbf{x}) = \mathbf{w}^\top\mathbf{x}$ for some $\mathbf{w} \in \mathbb{R}^n$.
$f(\mathbf{x}+\mathbf{x}') = \mathbf{w}^\top(\mathbf{x}+\mathbf{x}') = \mathbf{w}^\top\mathbf{x} + \mathbf{w}^\top\mathbf{x}' = f(\mathbf{x}) + f(\mathbf{x}')$
$f(\alpha\mathbf{x}) = \mathbf{w}^\top(\alpha\mathbf{x}) = \alpha\mathbf{w}^\top\mathbf{x} = \alpha f(\mathbf{x})$
Level set: The level set of a function $f : \mathbb{R}^n \to \mathbb{R}$ is a one-parametric family of sets defined as:
$L_f(c) \coloneqq \{\mathbf{x} : f(\mathbf{x}) = c\} = f^{-1}(c) \subseteq \mathbb{R}^n$
Level set of linear functions: Let $f : \mathbb{R}^n \to \mathbb{R}$ be linear, $f(\mathbf{x}) = \mathbf{w}^\top\mathbf{x} + b$, then
$L_f(c) = \{\mathbf{x} : \mathbf{w}^\top\mathbf{x} = c - b\}$ = hyperplane $\perp \mathbf{w}$
Composition of linear maps: Let $F_1, \dots, F_L$ be linear maps, then $F = F_L \circ \cdots \circ F_1$ is also a linear map.
Shows that every L-level hierarchy collapses to one level. ⇒ Need to move beyond linearity.
"Modest" generalization of linear maps?: Keep level set structure of linear function. *Rectified units* → piecewise linear functions, map is linear on each piece (polyhedra), continuous but typically non-differentiable (at border faces). *Generalized linear units* → invertible non-linearity, e.g. sigmoid functions, typically C^∞ (smooth).

2 Approx. Theory, Rectification, Sigmoid Nets

2.1 Ridge Functions

Definition: $f : \mathbb{R}^n \to \mathbb{R}$ is a ridge function, if it can be written as $f(\mathbf{x}) = \sigma(\mathbf{w}^\top\mathbf{x} + b)$ for some choice of $\sigma : \mathbb{R} \to \mathbb{R}, \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}$.
Denote the linear part of f by $\bar{f}(\mathbf{x}) = \mathbf{w}^\top\mathbf{x} + b$, then
$L_f(c) = \bigcup_{d \in \sigma^{-1}(c)} L_{\bar{f}}(d)$
if σ is differentiable at $z = \mathbf{w}^\top\mathbf{x} + b$, then
$\nabla_{\mathbf{x}} f \overset{\text{chain rule}}{=} \sigma'(x)\nabla_{\mathbf{x}}\bar{f} \overset{\text{directly}}{=} \sigma'(x)\mathbf{w}$
Theorem: Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable at $\mathbf{x}$. Then either $\nabla f(\mathbf{x}) = 0$ or $\nabla f(\mathbf{x}) \perp L_f(f(\mathbf{x}))$.
Pancake metaphor: Each pancake slice → same value function, same level set. The pancakes correspond to w_i, stacked to create $\mathbf{w}$.
Advantages of ridge functions: choice of *direction of change* is done in linear part ⇒ essentially equivalent to linear case. Non-linear activation function (σ): models the *rate of change* in the chosen direction. Just a $C(\mathbb{R})$ function, dimension independent. Continuous activation functions can be approximated by expansions with fixed activation function. Simplification at the cost of increased layer width. Therefore: continuous functions can be well-approximated by linear combinations of ridge functions (UAT).
Dense Approximations: A function class $H \subseteq C(\mathbb{R}^d)$ is dense in $C(\mathbb{R}^d)$ iff: $\forall f \in C(\mathbb{R}^d), \forall \epsilon > 0, \forall K \subset \mathbb{R}^d$, compact:
$\exists h \in H$ s.t. $\max_{x \in K} |f(\mathbf{x}) - h(\mathbf{x})| = ||f - h||_{\infty,K} < \epsilon$
Informally speaking: we can approximate any continuous f to arbitrary accuracy (on K) with a suitable member of H.
Approximation theorem: Let $\sigma \in C^\infty(\mathbb{R})$, not a polynomial, then $\mathcal{H}^1_\sigma$ is dense in $C(\mathbb{R})$.
Corollary: MLPs with one hidden layer and any non-polynomial, smooth activation function are universal function approximators.
Lemma: MLPs with one hidden layer and a polynomial activation function are **not** universal function approximators.
Remark: Smoothness requirement can be substantially weakened.

2.2 Rectification Networks

Rectified Linear Units: Activation function of ReLU is defined as
$(x)_+ \coloneqq \max(0, x)$, $\underbrace{\partial(x)_+}_{\text{subdifferential}} = \begin{cases} \{1\} & \text{if } x > 0 \\ \{0\} & \text{if } x < 0 \\ [0;1] & \text{if } x = 0 \end{cases}$
Absolute Value Rectification: Definition
$|x| \coloneqq \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise} \end{cases}$, $\partial|x| = \begin{cases} 1 & \text{if } x > 0 \\ [-1;1] & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$
Relation to ReLU activation: $(x)_+ = \frac{x+|x|}{2}$ and $|x| = 2(x)_+ - x$
Theorem: Networks with one hidden layer of ReLU or absolute value units are universal function approximators.
Question: by linearly combining m rectified units, into how many (= $R(m)$) cells is $\mathbb{R}^n$ maximally partitioned? $R(m) \leq \sum_{i=0}^{min(m,n)} = \frac{m!}{(m-i)!i!} = \binom{m}{i}$
Question: process n inputs through L ReLU layers with widths $m_1, ..., m_L \in O(m)$. Intro how many cells can $\mathbb{R}^n$ be maximally partitioned. Theorem (Montufar et al. 2014): it holds asymptotically that: $R(m, L) \in \Omega\big((\frac{m}{n})^{n(L-1)} m^n\big)$
Essentially, for any fixed n, exponential growth can be ensured by making layers sufficiently wide ($m > m$) and increasing the level of functional nesting (i.e. depth L).
Hinging Hyperplanes: Hinge function definition: If $g : \mathbb{R}^n \to \mathbb{R}$ can be written with parameters $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^n$ and $b_1, b_2 \in \mathbb{R}$ as below it is called a hinge function. $g(\mathbf{x}) = \max(\mathbf{w}_1^\top\mathbf{x} + b_1, \mathbf{w}_2^\top\mathbf{x} + b_2)$
Two hyperplanes, "glued" together at the face $(\mathbf{w}_1 - \mathbf{w}_2)^\top\mathbf{x} + (b_1 - b_2) = 0$
k-Hinge Functions: $g(\mathbf{x}) = \max(\mathbf{w}_1^\top\mathbf{x} + b_1, \dots, \mathbf{w}_k^\top\mathbf{x} + b_k)$
Theorem: Every continuous PWL function from $\mathbb{R}^n \to \mathbb{R}$ can be written as a signed sum of k-Hinges with $k \leq log_2[n+1]$. ⇒ $\sum_i \theta_i g_i(\mathbf{x}), \theta_i \in \pm 1$. Reducing the growth of absolute value nesting to logarithmic growth, instead of linear. PWL functions are dense and can approximate any function to any desired degree. This representation is exact (not approximate). Re-discovered by Goodfellow et al, 2013 as *Maxout*.
Wang's Theorem (2004): Every continuous piecewise linear function f can be written as the difference between two polyhedral functions. Explicity: there exist finite $\mathcal{A}^+, \mathcal{A}^-$ such that
$f(\mathbf{x}) = \max_{(\mathbf{w},b) \in \mathcal{A}^+} \{\mathbf{w}^\top\mathbf{x} + b\} - \max_{(\mathbf{w},b) \in \mathcal{A}^-} \{\mathbf{w}^\top\mathbf{x} + b\}$
Used by Goodfellow, 2013 to prove that a linear network with two maxout units (and a linear output unit, subtraction) are universal function approximators (exactly!). Performs well for speech recognition, beating sigmoid and ReLU in 2013. Higher improvement on smaller datasets, faster training.

2.3 Sigmoid Networks

Sigmoid activation function logistic function (or tanh).
$\sigma(t) = \frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t} \in (0;1)$, $\sigma^{-1}(\mu) = \ln\frac{\mu}{1-\mu}$
$\tanh(t) = 2\sigma(2t) - 1 \in (-1;1)$

3 Feedforward Networks

3.1 Feedforward Networks

Definition: a set of computational units arranged in a DAG. If, in contrast, output of network is fed back into the computation, we have a recurrent network. NN implements map $F : \mathbb{R}^n \to \mathbb{R}^m$. Compositional structure (layers): $F = F^L \circ F^{L-1} \circ \cdots \circ F^1$.
Linear + activation function: $F^l = \sigma^l \circ \bar{F}^l$, $\bar{F}^l(\mathbf{x}) = \mathbf{W}^l\mathbf{x} + \mathbf{b}^l$, $l = 1, \dots, L$.
"F minus output layer non-linearity": $\bar{F} = \bar{F}^L \circ \bar{F}^{L-1} \circ \cdots \circ \bar{F}^1$.

3.2 Output Units and Objectives

Loss function: is a non-negative function
$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$, $(\mathbf{y}^*, \mathbf{y}) \mapsto \ell(\mathbf{y}^*, \mathbf{y})$ such that
$\ell(\mathbf{y}, \mathbf{y}) = 0$ $(\forall \mathbf{y} \in \mathcal{Y})$ and $\ell(\mathbf{y}^*, \mathbf{y}) > 0$ $(\forall \mathbf{y} \neq \mathbf{y}^*)$ with output space $\mathcal{Y}$ ($\mathbf{y}^*$ is truth and $\mathbf{y}$ predicted).
Squared-error:
$\mathcal{Y} = \mathbb{R}^m$, $\ell(\mathbf{y}^*, \mathbf{y}) = \frac{1}{2}\|\mathbf{y}^* - \mathbf{y}\|_2^2 = \frac{1}{2}\sum_{i=1}^m (y_i^* - y_i)^2$
Classification error:
$\mathcal{Y} = [1 : m]$, $\ell(\mathbf{y}^*, \mathbf{y}) = 1 - \delta_{\mathbf{y}^*\mathbf{y}}$ with $\delta_{ab} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$
SVM loss: $max(0, 1-y)$ // Square loss $(1-y)^2$ // log regression: $log(1 + exp(-y))$
Expected Risk: The expected risk of F is given by
$\mathcal{R}^*(F) = \mathbf{E}_{\mathbf{x},\mathbf{y}}[\ell(\mathbf{y}, F(\mathbf{x}))]$ (cannot evaluate the functional $\mathcal{R}^*$ directly as the distribution governing inputs and outputs is generally unknown)
Training Risk: The expected risk under the empirical distribution induced by the sample $\mathcal{S}_N$. Assume we have a random sample of N input-output pairs, $\mathcal{S}_N \coloneqq \{(\mathbf{x}_i, \mathbf{y}_i) \overset{\text{i.i.d.}}{\sim} p : i = 1, \dots, N\}$.
The training risk (=empirical risk) of F on a training sample is defined as:
$\mathcal{R}(F; \mathcal{S}_N) = \frac{1}{N}\sum_{i=1}^N \ell(\mathbf{y}_i, F(\mathbf{x}_i))$
To minimize it, you just take the argmin over F. But watch out for **consistency**! **Regularization** is one way of enforcing consistency (which also makes sure you don't overfit to the data).
Probability distributions as outputs: Think of functions F as mappings from inputs to distributions $\mathcal{P}(\mathcal{Y})$ over outputs $\mathbf{y} \in \mathcal{Y}$.
$F : \mathbb{R}^n \to \mathbb{R}^m$, $\mathbf{x} \mapsto \mu$, $\mu \overset{\text{fixed}}{\mapsto} p(\mathbf{y}; \mu) \in \mathcal{P}(\mathcal{Y})$, $\mathbf{y} \sim p(\mathbf{y}; \mu)$
Each F effectively defines a conditional probability distribution (or conditional pdf) via $p(\mathbf{y}|\mathbf{x}; F) = p(\mathbf{y}; \mu = F(\mathbf{x}))$
Example: Multivariate Normal Distribution: μ is the mean of a normal distribution (with fixed covariance $\gamma^2\mathbf{I}$)
$p(\mathbf{y}|\mathbf{x}; F) = [\frac{1}{\sqrt{2\pi}\gamma}]^m \exp[-\frac{1}{2\gamma^2}\|\mathbf{y} - F(\mathbf{x})\|^2]$
so that: $-\log p(\mathbf{y}|\mathbf{x}; F) = C(\gamma) + \frac{1}{2\gamma^2}\|\mathbf{y} - F(\mathbf{x})\|^2$, which is equivalent to the squared error loss.
Generalized Linear Models: predict the mean of the output distribution. Use the form: $\mathbf{E}[\mathbf{y}|\mathbf{x}] = \sigma(\mathbf{w}^\top\mathbf{x})$, where σ is invertible and σ^{-1} is called the *link function*. Can be extended to also predict variances or dispersions.
Example: Logistic Regression
$\mathcal{Y} = \{0, 1\}$, $\mathcal{P}(\mathcal{Y}) = [0; 1]$, $\sigma(x) = 1/(1 + e^{-x})$, then:
$\mathbf{E}[y|\mathbf{x}] = p(1|\mathbf{x}) = \sigma(\mathbf{x}^\top\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top\mathbf{x}}}$
Link function: logit $\sigma^{-1}(t) = \log\Big(\frac{t}{1-t}\Big)$, $t \in (0; 1)$
For multinomial logistic regression: $\mathcal{Y} = [1 : m]$, $\mathcal{P}(\mathcal{Y})$ can be represented via soft-max
$p(y|\mathbf{x}) = \frac{\exp[z_y]}{\sum_{i=1}^m \exp[z_i]}$, $z_i \coloneqq \mathbf{w}_i^\top\mathbf{x}$, $i = 1, \dots, m$
over-parametrized model: set $\mathbf{w}_1 = \mathbf{0}$, s.t. $z_1 = 0$ (w.l.o.g.). Generalizes (binary) logistic regression (see exercises).
In neural networks: *non-linear* functions replace linear functions. Output layer units implement *inverse link function.*
Normal model (linear output layer):
$\mathbf{E}[\mathbf{y}|\mathbf{x}] = \bar{F}(\mathbf{x}) = \mathbf{W}^L(F^{L-1} \circ \cdots \circ F^1)(\mathbf{x}) + \mathbf{b}^L$
Logistic regression (sigmoid output layer):
$\mathbf{E}[y|\mathbf{x}] = \sigma(\bar{F}(\mathbf{x}))$
Logistic Log-Likelihood: Using shorthand $z \coloneqq \bar{F}(\mathbf{x}) \in \mathbb{R}$ then
$-\log p(y|z) = -\log\sigma((2y-1)z) = \zeta((1-2y)z$,
where $\zeta = \log(1 + \exp(\cdot))$ ζ is the *soft-plus/cross entropy loss* function.
Notes from this slide: cross entropy is a more general term though. Softplus is CE when your model is a Bernouilli. In information theory, KL is referred to as the relative entropy, $E_{P_{data}}[log(P_{data}]$ is the entropy and $-E_{P_{data}}[log(P_{model}]$ is the CE.

$KL_D(P_{data}||P_{model}) = E_{P_{data}}[log(P_{data}) - log(P_{model}))] = -E_{P_{data}}[log(P_{model})] + E_{P_{data}}[log(P_{data})]$

Multinomial log-likelihood: more generally (multinomial logistic regression), use shorthand: $\mathbf{z} \coloneqq \bar{F}_i(\mathbf{x}) \in \mathbb{R}^n$ then:

$$\mathcal{R}(F;(\mathbf{x},y)) = -\log p(y|\mathbf{x};F) = -\log\left[\frac{e^{z_y}}{\sum_{i=1}^m e^{z_i}}\right]$$

$$= -z_y + \underbrace{\log\sum_{i=1}^m \exp[z_i]}_{\text{log-partition fct.}} = \log\left[1+\sum_{i\neq y}\exp[z_i - z_y]\right]$$

3.3 Regularization

Weight decay is also known as ridge regression, logarithm of gaussian prior (MAP), maximum margin (SVM) and is related to early stopping and training with noise. **Other regularization approaches:** data augmentation, noise robustness (adding it to the input/weights/labels), semisupervised learning, multitask learning, early stopping, parameter sharing, ensembles, dropout.

4 Backpropagation

Gradient of objective with regard to parameters θ: $\nabla_\theta \mathcal{R} = (\frac{\partial \mathcal{R}}{\partial \theta_1}, \dots, \frac{\partial \mathcal{R}}{\partial \theta_d})^\top$

Steepest descent and *stochastic gradient descent*: $\theta(t+1) \leftarrow \theta(t) - \eta \nabla_\theta \mathcal{R}(\mathcal{S})$

With $t = 0, 1, 2, \dots$ is iteration index. $\mathcal{S}$ = all training data $\Rightarrow$ steepest descent; $\mathcal{S}$ = mini batch of data $\Rightarrow$ SGD

Basic steps for backprop: 1) forward pass for given training input $\mathbf{x}$ to compute activations for all units. 2) compute gradient of $\mathcal{R}$ w.r.t. output layer activations for given target $\mathbf{y}$. 3) iteratively propagate activation gradient information from outputs to inputs. 4) compute local gradients of activations w.r.t. weights.

Use chain rule: $(f \circ g)' = (f' \circ g) \cdot g'$.

Equivalently: $\frac{d(f\circ g)}{dx}\big|_{x=x_0} = \frac{df}{dz}\big|_{z=g(x_0)} \cdot \frac{dg}{dx}\big|_{x=x_0}$

Jacobi Matrix: vector valued function (map) $F : \mathbb{R}^n \to \mathbb{R}^m$. Each component function has gradient $\nabla F_i \in \mathbb{R}^n$, $i \in [1:m]$. Collect all gradients (as rows) into *Jacobi matrix* $(\mathbf{J}_F)_{ij} = \partial F_i / \partial x_j$:

$$\mathbf{J}_F \coloneqq \begin{bmatrix} \nabla^\top F_1 \\ \nabla^\top F_2 \\ \vdots \\ \nabla^\top F_m \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \frac{\partial F_m}{\partial x_2} & \cdots & \frac{\partial F_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m\times n}$$

Jacobi Matrix Chain Rule: With vector-valued functions $G : \mathbb{R}^n \to \mathbb{R}^q$, $H : \mathbb{R}^q \to \mathbb{R}^m$, $F \coloneqq H \circ G$:

$$\mathbf{J}_{H\circ G}\big|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{J}_H\big|_{\mathbf{z}=G(\mathbf{x}_0)} \cdot \mathbf{J}_G\big|_{\mathbf{x}=\mathbf{x}_0}$$

Deep function compositions: Notation! $\to$ index of a layer: superscript; index of a dimension of a vector: subscript; shorthand for layer activations $\mathbf{x}^l \coloneqq (F^l \circ \cdots \circ F^1)(\mathbf{x}) \in \mathbb{R}^{m_l}$ and $x_i^l \in \mathbb{R}$: activation of i-th unit in layer l; index of a data point, omitted when possible, rectangular brackets $(\mathbf{x}[i], \mathbf{y}[i])$.

Composition of multiple maps with a final cost function:

$$F = F^L \circ \cdots \circ F^1 : \mathbb{R}^n \to \mathbb{R}^m$$

$$\mathbf{x} = \mathbf{x}^0 \overset{F^1}{\mapsto} \mathbf{x}^1 \overset{F^2}{\mapsto} \mathbf{x}^2 \cdots \overset{F^L}{\mapsto} \mathbf{x}^L = \mathbf{y} \overset{\mathcal{R}}{\mapsto} \mathcal{R}(\theta;\mathbf{y})$$

Activity Backpropagation: Compute *activity gradients* in backward order via successive multiplication with Jacobians. Backpropagation of error terms $\mathbf{e}^l$. Effectively a linear network in reversed direction (w.r.t. the original) with "activities" $\mathbf{e}^l$.

$$\mathbf{e}^L \coloneqq \nabla_{\mathbf{y}}^\top \mathcal{R},\ \mathbf{e}^l \coloneqq \nabla_{\mathbf{x}^l}^\top \mathcal{R} = \mathbf{e}^L \cdot \mathbf{J}_{F^L} \cdots \mathbf{J}_{F^{l+1}} = \mathbf{e}^{l+1} \cdot \mathbf{J}_{F^{l+1}}$$

Jacobian for Ridge Functions: assuming differentiability of σ

$$\mathbf{x}^l = F^l(\mathbf{x}^{l-1}) = \sigma(\mathbf{W}^l\mathbf{x}^{l-1} + \mathbf{b}^l)$$

Hence: $\frac{\partial x_i^l}{\partial x_j^{l-1}} = \sigma'(\langle \mathbf{w}_i^l, \mathbf{x}^{l-1}\rangle + b_i^l) w_{ij}^l =: \tilde{w}_{ij}^l$ Thus: $\mathbf{J}_{F^l} = \tilde{\mathbf{W}}^l$

For ReLU $\tilde{w}_{ij}^l \in \{0, w_{ij}^l\} \Rightarrow \tilde{\mathbf{W}}^l$ = sparsified weight matrix

Quadratic Loss: $-\nabla_{\mathbf{y}}\mathcal{R}(\mathbf{x},\mathbf{y}^*) = -\nabla_{\mathbf{y}}\frac{1}{2}\|\mathbf{y}^* - \mathbf{y}\|^2 = \mathbf{y}^* - \mathbf{y}$

Multivariate Logistic Loss:

$$-\frac{\partial \mathcal{R}(\mathbf{x},y^*)}{\partial z_y} = \frac{\partial}{\partial z_y}\left[z_{y^*} - \log\sum_i \exp[z_i]\right] = \delta_{yy^*} - \frac{\exp[z_y]}{\sum_i \exp[z_i]} = \delta_{yy^*} - p(y|\mathbf{x})$$

Activations to Weights: Apply chain rule one more time: locally. Each weight/bias influences exactly one unit.

$$\frac{\partial \mathcal{R}}{\partial w_{ij}^l} = \frac{\partial \mathcal{R}}{\partial x_i^l} \cdot \frac{\partial x_i^l}{\partial w_{ij}^l} = \underbrace{\frac{\partial \mathcal{R}}{\partial x_i^l}}_{\text{backprop}} \cdot \underbrace{\sigma'(\langle \mathbf{w}_i^l, \mathbf{x}^{l-1}\rangle + b_i^l)}_{\text{sensitivity of } i\text{-th unit}} \cdot \underbrace{x_j^{l-1}}_{\substack{j\text{-th unit} \\ \text{activity}}}$$

5 CNN

Integral operator: Transform expressible with kernel H s.t. for any function f (for which Tf exists): $(Tf)(u) = \int_{t_1}^{t_2} H(u,t)f(t)dt$ (eg: Fourier transform)

Convolution: Given two functions f, h, their convolution is defined as:

$$(f * h)(u) \coloneqq \int_{-\infty}^{\infty} h(u-t)f(t)dt = \int_{-\infty}^{\infty} f(u-t)h(t)dt$$

Integral operator with kernel $H(u,t) = h(u-t)$, shift-invariant as $H(u-s, t-s) = h(u-t) = H(u,t)$ $(\forall s)$, commutative, typical use: f signal, h fast decaying kernel function.

To generalize to higher dimensions: Replace vectors by matrices (for discrete case), tensors for 3D and higher.

$$(F * G)[i,j] = \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} F[i-k, j-l] \cdot G[k,l]$$

Linear transform T is linear, if for all functions f, g and scalars α, β:

$$T(\alpha f + \beta g) = \alpha Tf + \beta Tg$$

Translation invariant transform T is translation/shift invariant, if for any f and scalar τ, $\quad f_\tau(t) \coloneqq f(t+\tau)$, $(Tf_\tau) = (Tf)(t+\tau)$

Any linear, translation-invariant transformation T can be written as a convolution with suitable h.

Discrete Cross-Correlation Vs Convolution: Let $f, h : \mathbb{Z} \to \mathbb{R}$, then:

$(h \star f)[u] \coloneqq \sum_{t=-\infty}^{\infty} h[t]f[u+t]$. aka "sliding inner product", $u+t$ vs $u-t$

Cross-correlation and convolution are closely related:

$$(h \star f)[u] = \sum_{t=-\infty}^{\infty} h[t]f[u+t] = \sum_{t=-\infty}^{\infty} h[-t]f[u-t]$$
$$= (\bar{h} * f)[u] = (f * \bar{h})[u], \text{ where } \bar{h}[t] \coloneqq h[-t]$$

The kernel h is "flipped over," which is non-commutative.

Toeplitz matrix: a matrix $\mathbf{H} \in \mathbb{R}^{k\prod n}$ is a Toeplitz matrix if there exists $n+k-1$ numbers $c_l (l \in [-(n-1):(k-1) \subset \mathbb{Z}$ such that: $H_{i,j} = c_{i-j}$. In plain English: all NW-SE diagonals are constant.

Bordering handling: padding with zeros $\to$ *same padding*. Only retain windows fully contained in support of signal f: $\to$ *valid padding*.

Backprop in Conv Layers: Exploit structural sparseness in computing $\frac{\partial x_i^l}{\partial x_j^{l-1}}$. Receptive field of $x_i^l : \mathcal{I}_i^l \coloneqq \{j : W_{ij}^l \neq 0\}$. (Where $\mathbf{W}$ is the Toeplitz matrix of the convolution). Obviously $\frac{\partial x_i^l}{\partial x_j^{l-1}} = 0$ for $j \notin \mathcal{I}_i^l$

Weight sharing in $\frac{\partial \mathcal{R}}{\partial h_j^l}$, where h_j^l is a kernel weight: $\frac{\partial \mathcal{R}}{\partial h_j^l} = \sum_i \frac{\partial \mathcal{R}}{\partial x_i^l}\frac{\partial x_i^l}{\partial h_j^l}$.

Weight is re-used for every unit within target layer $\to$ additive combination of derivatives in chain rule.

$$\frac{\partial L}{\partial w_{u,v}} = \sum_i\sum_j \frac{\partial L}{\partial y_{i,j}^{(l)}}\frac{\partial y_{i,j}^{(l)}}{\partial w_{u,v}} = \sum_i\sum_j \frac{\partial L}{\partial y_{i,j}^{(l)}}\frac{\partial}{\partial w_{u,v}}\Big(\sum_s\sum_t y_{i-s,j-t}^{(l-1)} w_{s,t}\Big) =$$
$$\sum_i\sum_j \frac{\partial L}{\partial y_{i,j}^{(l)}} y_{i-u,j-v}^{(l-1)} = \sum_i\sum_j \frac{\partial L}{\partial y_{i,j}^{(l)}} \text{rot}_{180} y_{u-i,v-j}^{(l-1)}$$

Convolution stages: the convolutional layer includes a convolution stage (affine transform), a detector stage (nonlinearity e.g. rectified linear) and a pooling stage (e.g. max pooling, average, soft-maximization). Only then do you move on to the next layer. **Convolutional Pyramid:** typical use of convoluton in vision: sequence of convolutions that reduce spatial dimensiosn (sub-sampling) and increase the number of channels.

1x1 'Convolution' as dimension reduction: For m channels of a $1\times 1\times k$ conv. $m \leq k : \mathbf{x}_{ij}^+ = \sigma(\mathbf{W}\mathbf{x}_{ij})$, $\mathbf{W} \in \mathbb{R}^{m\times k}$

Computing the size of the output image after applying a kernel: We apply m $f\text{x}f$ filters to an $n \times n \times d$ image, padding p, stride s. For each dimension of the image: $L = \frac{n+2p-f}{s} + 1$ Depth if going to be equal to the number m of filters. In general, setting zero padding to be $p = (f-1)/2$ when the stride is $s = 1$ ensures that the input volume and output volume will have the same size spatially. **With parameter sharing,** it introduces $f \cdot f \cdot d$ weights per filter, for a total of $(f \cdot f \cdot d) \cdot K$ weights and K biases.

6 Optimization

ML uses optimization, but is not equal to optimization.

6.1 Optimization for Deep Networks

Gradient Descent: Full gradient across all parameters

$\theta(t+1) \leftarrow \theta(t) - \eta\nabla_\theta\mathcal{R}(\mathcal{S})$ with $\eta > 0$ step size/learning rate.

Continuous time version (aka gradient flow, Euler's method): $\dot{\theta} = -\nabla_\theta\mathcal{R}$

Classic analysis: convex objective $\mathcal{R}$. $\mathcal{R}$ is minimum and θ^* is minimizer, i.e. $\mathcal{R}(\theta^*) \leq \mathcal{R}(\theta)$. $\mathcal{R}$ has L-Lipschitz-continuous gradients:

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

$\mathcal{R}$ is μ-strongly convex in θ (exponential convergence):

$\mathcal{R}(\theta(t)) - \mathcal{R}^* \leq \left(1 - \frac{\mu}{L}\right)^t(\mathcal{R}(\theta(0)) - \mathcal{R}^*)$

Rate of convergence depends adversely on condition number $\frac{L}{\mu}$.

Challenges: curvature may require to use small step sizes. $\mathbf{H}$ is the Hessian matrix.

$$\mathcal{R}(\theta - \eta\nabla\mathcal{R}) \overset{\text{Taylor}}{\approx} \mathcal{R}(\theta) - \eta\|\nabla\mathcal{R}\|^2 + \frac{\eta^2}{2}\underbrace{\nabla\mathcal{R}^\top\mathbf{H}\nabla\mathcal{R}}_{=\|\nabla\mathcal{R}\|_{\mathbf{H}}^2}$$

Problematic ill-conditioning: $\frac{\eta}{2}\|\nabla\mathcal{R}\|_{\mathbf{H}}^2\|\nabla\mathcal{R}\|^2$ (remedy for first order methods are small step sizes η).

Neural network cost functions can have many local minima and/or saddle points - and this is typical. Gradient descent can get stuck.

6.2 SGD

Choose update direction $\mathbf{v}$ at random such that $\mathbb{E}[\mathbf{v}] = -\nabla\mathcal{R}$, i.e. randomization scheme is unbiased. Pick random subset $\mathcal{S}_K \subseteq \mathcal{S}_N, K \leq N$. Note that $\mathbb{E}\mathcal{R}(\mathcal{S}_K) = \mathcal{R}(\mathcal{S}_N) \Rightarrow \mathbb{E}\nabla\mathcal{R}(\mathcal{S}_K) = \nabla\mathcal{R}(\mathcal{S}_N)$. SGD update step (randomization at each t): $\quad \theta(t+1) = \theta(t) - \eta(t)\nabla\mathcal{R}(t),\ \mathcal{R}(t) \coloneqq \mathcal{R}(\mathcal{S}_K(t))$

Epoch: one sweep through the data. Hard to analyze theoretically, works better in practice. No permutation of data $\Rightarrow$ danger of "unlearning".

Mini-batch size $k = 1$ often most efficient in terms of # backprop steps, but larger k better for utilizing concurrency in GPUs or multicore CPUs.

Rates: Under certain conditions SGD converges to the optimum: convex or strongly convex objective, Lipschitz gradients, decaying learning rate $(\sum_{t=1}^\infty \eta^2(t) < \infty,\ \sum_{t=1}^\infty \eta(t) = \infty$, typically $\eta(t) = Ct^{-\alpha},\ \frac{1}{2} < \alpha \leq 1)$, iterate (Polyak) averaging. Convergence rates: $O(1/t)$ suboptimality rate for strongly-convex case, $O(1/\sqrt{t})$ for non-strongly convex case.

SGD with momentum: accumulates the gradient over several updates, as a ball would accumulate speed. **Formula from HS2017:**

$$\theta(t+1) = \theta(t) - \eta\nabla\mathcal{R} + \alpha\underbrace{(\theta(t) - \theta(t-1))}_{\text{momentum}},\ \alpha \in [0;1)$$

Update the momentum: $m(t) = \alpha m(t-1) - (1-\alpha)\nabla_\theta\mathcal{R}(\theta(t-1)), \alpha, 1$

Bias correction: $\hat{m}(t) = \frac{m(t)}{(1-\alpha^t)}$ Update θ: $\theta(t) = \theta(t-1) + \eta\hat{m}(t)$

Nesterov momentum: compute the gradient where the momentum pushes you and make a correction. The gradient is evaluated at $\theta(t-1)+\eta\alpha\hat{m}(t-1)$.

$v(t) = \alpha v(t-1) - \epsilon\nabla_\theta\mathcal{R}(\theta(t-1) + \eta\alpha\hat{m}(t-1)), \alpha < 1$

AdaGrad: Intuitively, learning rate decays faster for weights that have seen significant updates. Consider the entire history of gradients, gradient matrix: $\theta \in \mathbb{R}^d, \mathbf{G} \in \mathbb{R}^{d\times t_{max}}, g_{it} = \frac{\partial\mathcal{R}(t)}{\partial\theta_i}\big|_{\theta=\theta(t)}$, compute (partial) row sums of $\mathbf{G}$

$$\gamma_i^2(t) \coloneqq \sum_{s=1}^t g_{is}^2 \text{ (note: not gradient norms)}$$

$$\theta_i(t+1) = \theta_i(t) - \frac{\eta}{\delta+\gamma_i(t)}\nabla\mathcal{R}(t),\ \delta > 0 \text{ (small)}$$

Non-convex variant (**RMSprop**): $\gamma_i^2 \coloneqq \sum_{s=1}^t \rho^{t-s}g_{is}^2,\ \rho < 1$

ADAM: AdaGrad + Momentum for SGD $(\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8})$

$g(t) = \nabla_\theta\mathcal{R}(\theta(t-1)$ (Get the gradient)

$m(t) = \beta_1 m(t-1) + (1-\beta_1)g(t)$ (Update the biased first moment estimate)

$v(t) = \beta_2 v(t-1) + (1-\beta_2)g(t)^2$ (Update the biased 2nd raw moment estimate)

$\hat{m}(t) = m(t)/(1-\beta_1^t)$ (Bias correction 1st moment estimate)

$\hat{v}(t) = v(t)/(1-\beta_2^t$ (Bias correction 2nd raw moment estimate)

$\theta(t) = \theta(t-1) + \eta\hat{m}(t)/(\sqrt{\hat{v}(t)} + \epsilon)$ (Update parameters)

6.3 Heuristics

Polyak averaging: average over iterates, instead of outputting the final iterate. Linear averaging (common for convex case): $\bar{\theta}(t) = \frac{1}{t}\sum_{s=1}^t \theta(s)$

Running averages (common for non-convex): $\bar{\theta}(t) = \alpha\theta(t-1) + (1-\alpha)\theta(t)$, time constant $\alpha \in [0,1]$

Batch Normalization: Normalize the layer activations and backpropagate through the normalization. Fix layer l, fix set of examples $I \subseteq [1:N]$. Mean activity, vector of standard deviation:

$$\mu_j^l := \frac{1}{|I|}\sum_{i\in I}(F_j^l \circ \cdots \circ F^1)(\mathbf{x}[i]) \in \mathbb{R}^{m_l}$$

$$\sigma_j^l := \sqrt{\delta + \frac{1}{|I|}\sum_i \Big((F_j^l \circ \cdots \circ F^1)(\mathbf{x}[i]) - \mu_j\Big)^2},\ \delta > 0$$

Normalized activities: $\tilde{\mathbf{x}}_j^l := \frac{\mathbf{x}_j^l - \mu_j}{\sigma_j}$,

To regain representational power: $\tilde{\tilde{\mathbf{x}}}_j^l = \alpha_j \tilde{\mathbf{x}}_j^l + \beta_j$

Can exactly undo the batch normalization, and directly control different learning dynamics, mean activation and scale. **Many more heuristics:** curriculum learning, continuation methods (define a family of simpler objective functions and track solutions), heuristics for initialization and pre-training.

6.4 Norm-Based Regularization

Regularization: Any aspect of a learning algorithm that is intended to lower the *generalization error* but not the training error. You add a functional Ω that is not dependent on the training data. Common choice:

L_2/Frobenius-norm penalty for deep networks:

$$\Omega(\theta) = \tfrac{1}{2}\sum_{l=1}^{L}\mu^l\|\mathbf{W}^l\|_F^2,\ \mu^l \geq 0 \text{ (common: penalize only weights, not biases)}$$

Weight Decay: Also based on L_2-norm

$\frac{\partial\Omega}{\partial w_{ij}^l} = \mu^l w_{ij}^l$, $\theta(t+1) = \underbrace{(1-\eta\mu)\cdot\theta(t)}_{\text{weight decay}} - \underbrace{\eta}_{\text{step size}} \cdot \underbrace{\nabla_\theta \mathcal{R}}_{\text{data dep.}}$ (assume $\mu^l = \mu$)

Weights in l-th layer get pulled toward zero with "gain". Naturally favors weights with small magnitude.

Analysis: Quadratic (Taylor) approx. of $\mathcal{R}$ around optimal θ^*:

$\mathcal{R}(\theta) \approx \mathcal{R}(\theta^*) + \frac{1}{2}(\theta-\theta^*)^\top \mathbf{H}(\theta-\theta^*)$, $\mathbf{H}_\mathcal{R} = (\frac{\partial^2\mathcal{R}}{\partial\theta_i\partial\theta_j})$, & $\mathbf{H} := \mathbf{H}_\mathcal{R}\big|_{\theta=\theta^*}$

First order optimality condition:

$$\begin{aligned}\nabla_\theta\mathcal{R}_\Omega \overset{!}{=} 0 &\Leftrightarrow \mathbf{H}(\theta-\theta^*) + \mu\theta = \mathbf{0}\\ &\Leftrightarrow (\mathbf{H}+\mu\mathbf{I})\theta = \mathbf{H}\theta^*\\ &\Leftrightarrow \theta = (\mathbf{H}+\mu\mathbf{I})^{-1}\mathbf{H}\theta^* = \mathbf{Q}\underbrace{(\mathbf{\Lambda}+\mu\mathbf{I})^{-1}\mathbf{\Lambda}}_{=\text{diag}(\frac{\lambda_i}{\lambda_i+\mu})}\mathbf{Q}^\top\theta^*\end{aligned}$$

with diagonalization $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$, $\mathbf{\Lambda} = \text{diag}(\lambda_1,\ldots,\lambda_d)$.

Interpretation: Along directions in parameter space with large eigenvalues of $\mathbf{H}$, i.e. $\lambda_i \gg \mu$: weight decay vanishes. Along directions in parameter space with small eigenvalues of $\mathbf{H}$, i.e. $\lambda_i \ll \mu$: weights decayed to nearly zero.

L1 Regularization: sparsity inducing choice:

$$\Omega(\theta) = \sum_{l-1}^{L}\mu^l\Big\|\mathbf{W}^l\Big\|_1 = \sum_{l=1}^{L}\mu^l\sum_{ij}|\mathbf{W}+ij^l|, \mu^l \geq 0$$

Using the subderivative, can derive the soft thresholding operator:

$w^* = \text{sign}(w_0)(\max\{|w_0|-\mu, 0\}$

6.5 Early Stopping/Dataset Augmentation

6.6 Dropout

Bagging: Ensemble method that combines models trained on bootstrap samples. **1)** Bootstrap sample $\tilde{\mathcal{S}}_N^k$: sample N times from $\mathcal{S}_N$ with replacement for $k = 1,\ldots,K$. **2)** Train model on $\tilde{\mathcal{S}}_N^k \to \theta_k$. **3)** Prediction: averaged model output probabilities $p(\mathbf{y}|\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K}p(\mathbf{y}|\mathbf{x};\theta^k)$

Dropout: Randomly "drop" subsets of units in network. Intuitively: this selects features which depend less on other features. Define "keep" probability π_i^l for unit i in layer l (typically 0.8 for input layers, 0.5 for hidden units). Realizes the ensemble: $p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{Z}}p(\mathbf{Z})p(\mathbf{y}|\mathbf{x};\mathbf{Z})$ where $\mathbf{Z}$ is the binary "zeroing" mask, ($p(\mathbf{Z})$ is defined via π_i^l). Then, you rescale each weight by π_i^l.

7 NLP

Word embeddings: Map symbols over vocabulary $\mathcal{V}$ to vector representation $\to$ embedding into a (Euclidean) vector space. Symbolic $\mathcal{V} \ni w \overset{\text{embed}}{\mapsto} \mathbf{x}_w \in \mathbb{R}^d$ quantitative. ($w_i \mapsto \mathbf{x}_i \in \mathbb{R}^D$)

Bilinear model: pairwise word co-occurence (pointwise mutual info):

$\text{pmi}(v,w) = \log\frac{p(v,w)}{p(v)p(w)} = \log\frac{p(v|w)}{p(v)} = \mathbf{x}_v^\top\mathbf{x}_w + \text{const}$

Skip-gram: Pairwise occurrences of words in context window of size R. Text is a long sequence of words, $\mathbf{w} = (w_1,\ldots,w_T)$. Co-occurrence index set: $\mathcal{C}_R := \{(i,j) \in [1:T]^2 : 1 \leq |i-j| \leq R\}$. Skip-gram objective function:

$\mathcal{L}(\theta;\mathbf{w}) = \sum_t\sum_{l=-c\neq 0}^{c}\log[p_\theta(w_{t+l}|w_t)]$. Model (soft-max):

$$\log p_\theta(v|w) = \mathbf{x}_v^\top\mathbf{z}_w - \log\underbrace{\sum_{u\in\mathcal{V}}\exp[\mathbf{x}_u^\top\mathbf{z}_w]}_{\text{partition function}}$$

In practice it's impractical to calculate the partition function. Use negative sampling and re-formulate relative word frequency $p(w)$ as logistic regression instead, $p_n(w) = p(w)^\alpha$:

$$\mathcal{L}(\theta;\mathbf{w}) = \sum_{(i,j)\in\mathcal{C}_\mathbb{R}}\Big[\underbrace{\log\sigma(\mathbf{x}_{w_i}^\top\mathbf{z}_{w_j})}_{\text{positive examples}} + k\mathbf{E}_{v\sim p_n}\underbrace{[\log\sigma(-\mathbf{x}_{w_i}^\top\mathbf{z}_v)]}_{\text{negative examples}}\Big]$$

From words to sequences of words: language models via embeddings

$\log p(v|\mathbf{w} := w_1,\ldots,w_{t-1}) = \mathbf{x}_v^\top\mathbf{z}_\mathbf{w} + \text{const}$, with $\mathbf{x}_v$ word embedding, $\mathbf{z}_\mathbf{w}$ sequence embedding.

Three approaches: 1) CNN: concepually simple, fast to train but limited range memory. **2) Recurrent networks**: active memory management via gated units but more difficult to optimize and larger datasets needed. **3) Recursive networks**: in combination with parsers.

7.1 Conv. Nets for Natural Language

Map word sequence to vector sequence, get sentence matrix. Concatenate vectors. Convolve (nonlinearity is usually tanh or ReLU). Pooling (reduce each channel of variable length to single number). Predict words via softmax. $p(v|\mathbf{w}) = \frac{\exp(\langle\mathbf{y}_v,\mathbf{z}_\mathbf{w}\rangle)}{\sum_{u\in\mathcal{V}}\exp(\langle\mathbf{y}_u,\mathbf{z}_\mathbf{w}\rangle)}$ uses word embeddings $\mathbf{y}_v \in \mathbb{R}^k$.

7.2 Recurrent Networks

Linear dynamical system: $\bar{F}(\mathbf{h},\mathbf{x};\theta) := \mathbf{W}\mathbf{h}+\mathbf{U}\mathbf{x}+\mathbf{b}$, $\theta = (\mathbf{U},\mathbf{W},\mathbf{b},\ldots)$

As always, compose with element-wise non-linearity:

$F := \sigma\circ\bar{F}$, $\sigma \in \{\text{logistic, tanh, ReLU},\ldots\}$

Optionally produce outputs: $\mathbf{y} = H(\mathbf{h};\theta) := \sigma(\mathbf{V}\mathbf{h}+\mathbf{c})$, $\theta = (\ldots,\mathbf{V},\mathbf{c})$

Loss functions, 2 settings: **output at end** ($t=T$): $\mathbf{h}^T \mapsto H(\mathbf{h}^T;\theta) = \mathbf{y}$

output at every time step: seq $(\mathbf{y}^1,\ldots,\mathbf{y}^T)$:

$\mathcal{R}(\mathbf{y}^1,\ldots,\mathbf{y}^T) = \sum_{t=1}^{T}\mathcal{R}(\mathbf{y}^t) = \sum_{t=1}^{T}\mathcal{R}(H(\mathbf{h}^t;\theta))$ (additive loss function)

Backprop through time (BPTT): Unroll network, define shortcut

$\dot{\sigma} := \sigma'(\bar{F}_i(h^{t-1},x^t))$, then

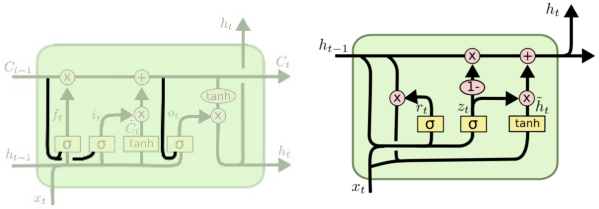
$$\frac{\partial\mathcal{R}}{\partial w_{ij}} = \sum_{t=1}^{T}\frac{\partial\mathcal{R}}{\partial h_i^t}\cdot\frac{\partial h_i^t}{\partial w_{ij}} = \sum_{t=1}^{T}\frac{\partial\mathcal{R}}{\partial h_i^t}\cdot\dot{\sigma}_i^t\cdot h_j^{t-1}$$

$$\frac{\partial\mathcal{R}}{\partial u_{ik}} = \sum_{t=1}^{T}\frac{\partial\mathcal{R}}{\partial h_i^t}\cdot\frac{\partial h_i^t}{\partial u_{ik}} = \sum_{t=1}^{T}\frac{\partial\mathcal{R}}{\partial h_i^t}\cdot\dot{\sigma}_i^t\cdot x_k^t \text{ (notice weight sharing)}$$

Exploding/Vanishing gradients: spectral norm of a matrix = largest singular value $\sigma_{\max}(\mathbf{A})$. If $\sigma_{\max}(\mathbf{W}) < 1$, gradients are vanishing, if $\sigma_{\max}(\mathbf{J}_F) > 1$, gradients may explode.

8 Memory & Attention

Long-term dependencies: LSTM with peepholes and **GRUs**:



$i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1})$ input gate

With peepholes: $i_t = \sigma(W_i\cdot[C_{t-1},h_{t-1},x_t]+b_i)$

$o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1})$ output gate

With peepholes: $o_t = \sigma(W_o\cdot[C_{t-1},h_{t-1},x_t]+b_o)$

$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1})$ forget gate

With peepholes: $f_t = \sigma(W_f\cdot[C_{t-1},h_{t-1},x_t]+b_t)$

$\tilde{c}_t = \tanh\Big(W^{(c)}x_t + U^{(c)}h_{t-1}\Big)$ preparing input to add to memory

$c_t = f_t\odot c_{t-1} + i_t\odot\tilde{c}_t$ updating memory (combine stored and new)

$h_t = o_t\odot\tanh(c_t)$ output

Where $\odot$ is point-wise multiplication between vectors.

Gated memory units::

$z_t = \sigma(W_z\cdot[h_{t-1},x_t])$ $\quad r_t = \sigma(W_r\cdot[h_{t-1},x_t])$

$\tilde{h}_t = tanh(W\cdot[r_t * h_{t-1},x_t])$ $\quad h_t = (1-z_t)*h_{t-1} + z_t*\tilde{h}_t$

8.1 Recursive Networks

Recurrent = linear chain structure, depth eff. $\mathbf{O}(n)$, $\mathbf{h}^t = F(\mathbf{h}^{t-1},\mathbf{x}^t)$

Recursive = tree structure, $\mathbf{O}(\log n)$, $\mathbf{h}^n = F(\mathbf{h}^{\text{n.left}},\mathbf{h}^{\text{n.right}})$.

Learn composition $F : \mathbb{R}^d\times\mathbb{R}^d \to \mathbb{R}^d$, then applied at each inner node of the tree. Application of recursive networks: sentiment analysis.

9 Autoencoders

9.1 Autoencoders

Given data points $\mathbf{x} \in \mathbb{R}^n, i = 1,\ldots,k$.

Goal: compress data into m-dimensional ($m \leq n$) representation.

Linear Autoencoder: $\mathbf{x} \overset{\mathbf{C}}{\mapsto} \mathbf{z} \overset{\mathbf{D}}{\mapsto} \hat{\mathbf{x}} \overset{\mathcal{R}}{\mapsto} \frac{1}{2}\|\mathbf{x}-\hat{\mathbf{x}}\|^2$, choose $\mathbf{C},\mathbf{D}$ s.t.

$$\frac{1}{2k}\sum_{i=1}^{k}\|\mathbf{x}_i - \mathbf{D}\mathbf{C}\mathbf{x}_i\|^2 \leftarrow \min$$

Optimal Linear Compression:

Proposition: Given data $\mathbf{X} = \mathbf{U}\cdot\text{diag}(\sigma_1,\ldots,\sigma_n)\cdot\mathbf{V}^\top$. The choice $\mathbf{C} = \mathbf{U}_m^\top$ and $\mathbf{D} = \mathbf{U}_m$ minimizes the squared reconstruction error of a two-layer linear auto-encoder with m hidden units. **Proof**:

$$\mathbf{D}\mathbf{C}\mathbf{X} = \underbrace{\mathbf{U}_m}_{\text{1st}}\underbrace{\mathbf{U}_m^\top}_{\text{2nd}}\underbrace{\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top}_{\text{data}} = \mathbf{U}_m[\mathbf{I}_m\ \ \mathbf{0}]\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{U}_m[\mathbf{\Sigma}_m\ \ \mathbf{0}]\mathbf{V}^\top$$
$$= \mathbf{U}_m\mathbf{\Sigma}_m\mathbf{V}_m^\top$$

Comment: note that we can do weight sharing between the decoder and the encoder networks: $\mathbf{D} = \mathbf{C}^\top$

Principal Component Analysis: Assume we have centered the data

$\mathbf{x}_i \mapsto \mathbf{x}_i - \frac{1}{k}\sum_{i=1}^{k}\mathbf{x}_i$

$\Rightarrow \mathbf{S} := \mathbf{X}\mathbf{X}^\top \in \mathbb{R}^{n\times n}$ is the sample covariance matrix

$\Rightarrow \mathbf{S} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top$

$\Rightarrow$ columns of $\mathbf{U}$ are eigenvectors of covariance matrix.

$\Rightarrow \mathbf{U}_m\mathbf{U}_m^\top$ is projection to m principcal components of $\mathbf{S}$

This is equivalent to PCA.

Regularized Autoencoder: Can use L2 norm (ability to learn "overcomplete" codes), L1 norm (code sparseness), or contractive autoencoders ($\Omega(\mathbf{z}) = \lambda\|\frac{\partial\mathbf{z}}{\partial\mathbf{x}}\|_F^2$ penalizes Jacobian and generalizes weight decay)

Denoising Autoencoder: Perturb inputs $\mathbf{x} \mapsto \mathbf{x}_\eta$, where $\boldsymbol{\eta}$ is a random noise vector (e.g. additive (white) noise). **Idea**: learn robust under noise features.

$\mathbf{x}_\eta = \mathbf{x} + \boldsymbol{\eta}$, $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0},\sigma^2,\mathbf{I})$, $\min \to \mathbb{E}_\mathbf{x}\mathbb{E}_{\boldsymbol{\eta}}[\ell(\mathbf{x},(H\circ G)(\mathbf{x}_{\boldsymbol{\eta}}))]$

Hope: $\|\mathbf{x}-\mathbf{x}_{\boldsymbol{\eta}}\|^2 > \|\mathbf{x}-H(G(\mathbf{x}_{\boldsymbol{\eta}}))\|^2 \Rightarrow$ de-noising

9.2 Factor Analysis

Latent Variable Models: Generic way of defining probabilistic i.e. generative models: latent variable models.

1) Latent variable $\mathbf{z}$, with distribution $p(\mathbf{z})$

2) Conditional models for observables $\mathbf{x}$, $p(\mathbf{x}|\mathbf{z})$

3) Observed data model: integrating/summing out latent variables:

$$p(\mathbf{x}) = \int p(\mathbf{z})p(\mathbf{x}|\mathbf{z})\mu(d\mathbf{z}) = \begin{cases}\mu = \text{ Lebesgue}, & \int p(\mathbf{z})p(\mathbf{x}|\mathbf{z})d\mathbf{z}\\ \mu = \text{ counting}, & \sum_\mathbf{z} p(\mathbf{z})p(\mathbf{x}|\mathbf{z})\end{cases}$$

Simple discrete model: mixture model

$\mathbf{z} \in \{1,\ldots,K\}$, $p(\mathbf{z}) =$ mixing proportions.

$p(\mathbf{x}|\mathbf{z})$: conditional densities (e.g. Gaussians)

Linear Factor Analysis: Latent variable *prior* $\mathbf{z} \in \mathbb{R}^m$, $\mathbf{z} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$

Linear *observation model*, $\mathbf{x} \in \mathbb{R}^n$

$\mathbf{x} = \boldsymbol{\mu} + \mathbf{W}\mathbf{z} + \boldsymbol{\eta}$, $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0},\mathbf{\Sigma})$, $\mathbf{\Sigma} := \text{diag}(\sigma_1^2,\ldots,\sigma_n^2)$

$\boldsymbol{\eta}$ and $\mathbf{z}$ are independent, typically: $m < n$ (fewer factors than features), few factors account for dependencies between many observables, MLE for μ on training set: $\hat{\boldsymbol{\mu}} = \frac{1}{k}\sum_{i=1}^{k}\mathbf{x}_i$ (can assume data is centered)

Proposition: in the factor analysis model $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu},\mathbf{W}\mathbf{W}^\top + \mathbf{\Sigma})$

Moment Generating Functions: of random vector $\mathbf{x}$:
$M_x : \mathbb{R}^n \to \mathbb{R}$, $M_\mathbf{x}(\mathbf{t}) := \mathbb{E}_\mathbf{x}\exp[\mathbf{t}^\top\mathbf{x}]$ $(= \int_{-\infty}^{\infty}\mathbf{t}^\top\bar{\mathbf{x}}\cdot f(\bar{\mathbf{x}})\,d\bar{\mathbf{x}})$
Uniqueness theorem: If $M_\mathbf{x}$, $M_\mathbf{y}$ exist for RVs $\mathbf{x},\mathbf{y}$ and $M_\mathbf{x} = M_\mathbf{y}$ then (essentially) $p(\mathbf{x}) = p(\mathbf{y})$.
$M_\mathbf{x}$ represents *moments* of $\mathbf{x}$ in the following way: $k_1,\dots,k_n \in \mathbb{N}$,
$\mathbb{E}[x^{k_1}\cdots x^{k_n}] = \frac{\partial^k}{\partial t_1^{k_1}\cdots\partial t_n^{k_n}} M_\mathbf{x}\big|_{\mathbf{t}=0}$
Moment generating functions can be used to deal with *sums of i.i.d. random variables*: $M_{\mathbf{x}+\mathbf{y}} = M_\mathbf{x}\cdot M_\mathbf{y}$
Multivariate Gaussian: $\boldsymbol{\Sigma}$ is variance-covariance matrix: $\mathbf{E}(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^\top$
$\text{PDF} = p(\mathbf{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{\exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]}{\sqrt{(2\pi)^\mathbf{n}\cdot\det(\boldsymbol{\Sigma})}}$
Moment generating function: $M_\mathbf{x}(\mathbf{t}) = \exp\left[\mathbf{t}^\top\boldsymbol{\mu} + \frac{1}{2}\mathbf{t}^\top\boldsymbol{\Sigma}\mathbf{t}\right]$

9.3 Deep Gaussian Models

Generalize factor analysis with *depth.*
Noise variables: $\mathbf{z}^l \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0},\mathbf{I})$, $l = 1,\dots,L$
Hidden activities (top-down $\mathbf{h}^L \to \mathbf{h}^1$:
$\mathbf{h}^L = \mathbf{W}^L\mathbf{z}^L,\ \mathbf{h}^l = \underbrace{F^l(\mathbf{h}^{l+1})}_{\text{deterministic}} + \underbrace{\mathbf{W}^l\mathbf{z}^l}_{\text{stochastic}}$
Hidden layer (conditional) distribution [vs factor analysis]
$\mathbf{h}|\mathbf{h}^+ \sim \mathcal{N}(F(\mathbf{h}^+),\mathbf{W}\mathbf{W}^\top)$ [vs. $\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu},\mathbf{W}\mathbf{W}^\top+\boldsymbol{\Sigma})$]
Generative Model Estimation: We assume that, for a given $\mathbf{x}$, $q(\mathbf{z}|\mathbf{x})$ is fixed (part of ELBO). How can we optimize θ?
Sample noise variables $(\mathbf{z}^1,\dots,\mathbf{z}^L) \sim q(\mathbf{z}|\mathbf{x})$. Perform backpropagation and SGD step for θ. If q is simple, sampling is efficient (no MCMC needed). Unbiased estimation of gradient, small variance overhead. Maybe even be beneficial (training with noise).
Stochastic Backpropagation: Optimizing over q involves gradients of expectations. How does a change of the q-distributions change q-averages?
$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$, f : smooth and integrable, then
$\nabla_{\boldsymbol{\mu}}\mathbb{E}[f(\mathbf{z})] = \mathbb{E}[\nabla_\mathbf{z} f(\mathbf{z})]$, $\nabla_{\boldsymbol{\Sigma}}\mathbb{E}[f(\mathbf{z})] = \frac{1}{2}\mathbb{E}[\nabla^2_\mathbf{z} f(\mathbf{z})]$
via integration by parts:
$\nabla_{\boldsymbol{\mu}}\mathbb{E}f(\mathbf{z}) = -\int\nabla_\mathbf{z}\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})f(\mathbf{z})d\mathbf{z} = \int\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})\nabla_\mathbf{z} f(\mathbf{z})d\mathbf{z}$
(used: $\int f(\mathbf{z})\nabla_{\boldsymbol{\mu}}p(\mathbf{z})d\mathbf{z} \overset{\text{Gaussian}}{=} -\int f(\mathbf{z})\nabla_\mathbf{z} p(\mathbf{z})d\mathbf{z}$)
$= \int\nabla_\mathbf{z} f(\mathbf{z})p(\mathbf{z})d\mathbf{z} - \underbrace{[f\cdot p]_{-\infty}^{\infty}}_{=0\text{ “sloppy”}} = \mathbb{E}[\nabla_\mathbf{z} f(\mathbf{z})]$
Deep Latent Gaussian Models: 2 coupled networks: top-down (generative) and bottom-up (recognition). Forward pass: deterministic recognition, sampled generative. Backward pass: deterministic, but: stochastic backprop.

10 Density Estimation & GANs

10.1 Density Estimation

Standard problem in statistics and unsupervised learning: learn the distribution of the data. Classically: parametric family of densities $p(\theta)$, $\theta\in\Theta$.
MLE: $\theta^* \overset{\max}{\to} \mathbb{E}[\ln p_\theta(\mathbf{x})]$, $\mathbf{x}\sim p(\mathbf{x})$. In practice: expectation w.r.t. empirical distribution
Prescribed models: Ensure that p_θ defines a proper density $\int p_\theta(\mathbf{x})d\mathbf{x} = 1$
Trivial for models like exponential families. Impractical for complex models (Markov nets, DNNs). What are strategies for more complex models?
Latent Variable Models: Classically: Define complex models via marginalization of a latent variable model $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x},\mathbf{z})d\mathbf{z}$ (or sum)
EM algorithm (ELBO: evidence lower bound)
$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_q[\log p_\theta(\mathbf{x},\mathbf{z})] + D_{\text{KL}}(q(\mathbf{z})||p_\theta(\mathbf{z})) \overset{\max\mathbb{E}}{\leftarrow} \theta, q$
optimal $q(\mathbf{x};\mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x})$ (posterior) not always tractable
Variational inference: further approximation. Restrict to simpler families of distribution (weakening of ELBO), amortized inference $\Rightarrow$ variational autoencoders.
Unnormalized models: Represent improper density functions:
$\underbrace{\bar{p}_\theta(\mathbf{x})}_{\text{represented}} = \underbrace{c_\theta}_{\text{unknown}}\cdot\underbrace{p_\theta(\mathbf{x})}_{\text{normalized}}$
Can't use log-likelihood as scaling of $\bar{p}_\theta$ will cause unbounded likelihood. Alternative estimation method for unnormalized models?
Score matching: Is there an operator we can apply to $\bar{p}_\theta$ which does not depend on normalization? Score matching: $\psi_\theta := \nabla\log\bar{p}_\theta$, $\psi = \nabla\log p$
Minimize criterion: $J(\theta) = \mathbb{E}\|\psi_\theta - \psi\|^2$
Equivalently (eliminate ψ using integration by parts):
$J(\theta) \overset{\pm c}{=} \mathbb{E}[\sum_i \partial_i\psi_{\theta,i} - \frac{1}{2}\psi_{\theta,i}^2]$ Expectation approximated by sampling.
Implicit Models: Statistical models via: *generating stochastic mechanism* or *simulation process*. **Deep implicit models:**
1) Latent code $\mathbb{R}^d \ni \mathbf{z}\sim\pi(\mathbf{z})$, e.g. $(\mathbf{z}) = \mathcal{N}(\mathbf{0},\mathbf{I})$ **2) parameterized mechanism** $F_\theta : \mathbb{R}^d \to \mathbb{R}^m$ **3) induced distribution** $\mathbb{R}^m \ni \mathbf{x}\sim p_\theta(\mathbf{x})$ **4) sampling** is easy: random vector + forward propagation
Noise Contrastive Estimation: *bootstrap* generative models via classification problems. Reduce density estimation to binary classification. Define probability (p_n: “contrastive” distribution: known distribution of negative samples, say “everything but dogs” and $p(\mathbf{x})$ is “dogs”)
$\tilde{p}(\mathbf{x},y=1) = \frac{1}{2}p(\mathbf{x})$, $\tilde{p}(\mathbf{x},y=0) = \frac{1}{2}p_n(\mathbf{x})$
Probabilistic classifier (induced by $\bar{p}_\theta$) $q_\theta = \frac{\alpha\bar{p}_\theta}{\alpha\bar{p}_\theta + p_n}$, $\alpha > 0$.
Bayes optimal if $\alpha\bar{p}_\theta = p$. Minimize logistic loss with regard to θ and $\alpha > 0$
This estimator is consistent! *Sometimes* statistically efficient, generally no.

10.2 Variational Auto Encoders

10.3 Generative Adversarial Models

Classification problem: distinguish between data & model.
$\tilde{p}_\theta(\mathbf{x},y=1) = \frac{1}{2}p(\mathbf{x})$, $\tilde{p}_\theta(\mathbf{x},y=0) = \frac{1}{2}p_n(\mathbf{x})$
Bayes optimal classifier: posterior $q_\theta = p/(p+p_\theta)$.
Train generator via minimizing the logistic likelihood:
$\theta \overset{\min}{\to} \ell^*(\theta) := \mathbb{E}_{\tilde{p}_\theta}[y\ln q_\theta(\mathbf{x}) + (1-y)\ln(1-q_\theta(\mathbf{x}))]$
Minimize Jensen-Shannon Divergence to get $\ell^* = \text{JS}(p,p_\theta) - \ln 2$
Generator's goal: generate samples that are *indistinguishable from real data*, even for the best possible classifier. In general this in inaccessible, so we define a classfication model: $q_\phi : \mathbf{x}\mapsto[0;1]$, $\phi\in\Phi$.
Define objective via bound:
$\ell^*(\theta) \geq \sup_{\phi\in\Phi}\ell(\theta,\phi)$, $\ell(\theta,\phi) := \mathbb{E}_{\tilde{p}_\theta}[y\ln q_\phi(\mathbf{x}) + (1-y)\ln(1-q_\phi(\mathbf{x}))]$
Find best classifier within restricted family. Typically: Φ = weight space of DNN. Training objective is defined implicitly over sup.
Optimizing GANs: Saddle point problem $\theta^* := \text{argmin}_{\theta\in\Theta}\{\sup_{\phi\in\Phi}\ell(\theta,\phi)\}$. Explicitly performing inner sup is impractical.
SGD as a heuristic (may diverge!):
$\theta^{t+1} = \theta^t - \eta\nabla_\theta\ell(\theta^t,\phi^t)$ $\phi^{t+1} = \phi + \eta\nabla_\phi\ell(\theta^{t+1},\phi^t)$
Evaluating GANs: Conceptual question: how to measure *quality* of implicit models? likelihood-based method not valid for implicit models. Trade-offs: noisy samples (e.g. blurry images), but adequate representation of variability. Faithful (as in: good looking) samples, but lack of representation (“mode dropping”). Which is better?

Matrix Things

SVD: Suppose $\mathbf{M}$ is a $m\times n$ matrix whose entries are real or complex numbers. There exists a factorization, called a SVD of $\mathbf{M}$, of the form $\mathbf{M} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*$ where **1)** $\mathbf{U}$ is an $m\times m$ unitary matrix over K (if $K=\mathbb{R}$, unitary matrices are orthogonal matrices), **2)** $\boldsymbol{\Sigma}$ is a diagonal $m\times n$ matrix with non-negative real numbers on the diagonal (singular values), **3)** $\mathbf{V}$ is an $n\times n$ unitary matrix over K, and **4)** $\mathbf{V}$ is the conjugate transpose of $\mathbf{V}$. A common convention is to list the singular values in descending order.
Matrix calculus
$(\mathbf{A}+\mathbf{B})' = \mathbf{A}'+\mathbf{B}'$, $(\mathbf{ABC})' = \mathbf{A}'\mathbf{BC}+\mathbf{AB}'\mathbf{C}+\mathbf{ABC}'$,
$(\mathbf{A}^n)' = \mathbf{A}'\mathbf{A}^{n-1}+\mathbf{A}\mathbf{A}'\mathbf{A}^{n-2}+\dots+\mathbf{A}^{n-1}\mathbf{A}'$,
$(\mathbf{A}^{-1})' = -\mathbf{A}^{-1}\mathbf{A}'\mathbf{A}^{-1}$, $(\det(\mathbf{A}))' = \text{Tr}(\mathbf{A}'\mathbf{A})$,
$(\mathbf{Ax})' = \mathbf{A}^\top$, $(\mathbf{x}^\top\mathbf{A})' = \mathbf{A}$, $(\mathbf{x}^\top\mathbf{x})' = 2\mathbf{x}$, $(\mathbf{x}^\top\mathbf{Ax})' = \mathbf{Ax}+\mathbf{A}^\top\mathbf{x}$ - $(\mathbf{AB})^\top = \mathbf{B}^\top\mathbf{A}^\top$
$\mathbf{A}^{-1} = \begin{bmatrix}a & b\\ c & d\end{bmatrix}^{-1} = \frac{1}{\det\mathbf{A}}\begin{bmatrix}d & -b\\ -c & a\end{bmatrix} = \frac{1}{ad-bc}\begin{bmatrix}d & -b\\ -c & a\end{bmatrix}$
$|A| = \begin{vmatrix}a & b & c\\ d & e & f\\ g & h & i\end{vmatrix} = a\begin{vmatrix}e & f\\ h & i\end{vmatrix} - b\begin{vmatrix}d & f\\ g & i\end{vmatrix} + c\begin{vmatrix}d & e\\ g & h\end{vmatrix}$
Trace identities and differentiation rules:
$\mathbf{v}^\top\mathbf{w} = \sum_i v_i w_i = \text{Tr}(\mathbf{A}+\mathbf{B})$ $\text{Tr}(\mathbf{A}+\mathbf{B}) = \text{Tr}(\mathbf{A}) + \text{Tr}(\mathbf{B})$
$\mathbf{E}\,\text{Tr}(\mathbf{X}) = \text{Tr}\,\mathbf{E}[\mathbf{X}]$ $\nabla_\mathbf{A}\,\text{Tr}\left(\mathbf{AA}^\top\right) = 2\mathbf{A}$, $\nabla_\mathbf{A}\,\text{Tr}(\mathbf{AB}) = \mathbf{B}^\top$
Definiteness of a matrix:
$\mathbf{A}$ is positive definite if $\mathbf{x}^\top\mathbf{Ax} > 0$, $\forall\mathbf{x}\neq 0$ $\mathbf{A}$ is indefinite if it contains positive and negative eigenvalues.
Finding Eigenvalues: $\det(\mathbf{A}-\lambda\mathbf{I}) = 0$ and find values of λ. To get eigenvector for λ_i, plug λ_i into $(\mathbf{A}-\lambda_i\mathbf{I})\mathbf{v} = 0$ and solve.
Roots of quadratic polynomial: $x = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$

General Math

Derivative rules
$\tan(x)\to\sec^2(x)$, $a^x\to\ln(a)a^x$, $f/g\to(f'g-g'f)/g^2$
Sigmoid Derivative: $\frac{\text{d}\sigma(x)}{\text{d}x} = \sigma(x)(1-\sigma(x))$

Hessian: when taking it, keep in mind that the second variable w.r.t which you take the the second partial derivative is what stays fixed per row.
Integration by parts: $\int u\,dv = uv - \int v\,du$
$\int_a^b u(x)v'(x)\,dx = [u(x)v(x)]_a^b - \int_a^b u'(x)v(x)dx$
$= u(b)v(b) - u(a)v(a) - \int_a^b u'(x)v(x)\,dx$
Integration by Substitution: $\int f(g(x))g'(x)dx\to\int f(u)du$
Fundamental Theorem of Calculus: $f(y) - f(x) = \int_x^y\nabla f(\tau)d\tau$
Mean: $\mathbb{E}[f(x)] = \int_x x\,f(x)\,dx$ **Variance:** $\mathbb{E}[f(x)] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2$
Convexity of f implies: $f(y)\geq f(x)+\nabla f(x)^\top(y-x), \forall(x,y)\in\mathbb{R}^{d\times d}$
Jensen's inequality: Let (Ω,A,μ) be a probability space, such that $\mu(\Omega) = 1$. If g is a real-valued function that is μ-integrable, and if φ is a convex function on the real line, then: $\varphi\left(\int_\Omega g\,d\mu\right)\leq\int_\Omega\varphi\circ g\,d\mu$.
Cauchy-Schwarz: $|\langle\mathbf{u},\mathbf{v}\rangle|^2\leq\langle\mathbf{u},\mathbf{u}\rangle\cdot\langle\mathbf{v},\mathbf{v}\rangle$
Taylor Expansion $f(x) = \sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$ (exp. around x_0).
$f(\mathbf{w})\approx f(\bar{\mathbf{w}}) + (\mathbf{w}-\bar{\mathbf{w}})^\top\nabla f(\bar{\mathbf{w}}) + \frac{1}{2}(\mathbf{w}-\bar{\mathbf{w}})^\top H(\mathbf{w}-\bar{\mathbf{w}})$
Taylor-Lagrange: $f(x) = \sum_{k=0}^{n}\frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + \int_{x_0}^{x}\frac{f^{(n+1)}(t)}{n!}(x-t)^n dt$.
Taylor expansion with Lagrange remainder:
$f(x) = \sum_{k=0}^{n}\frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + \frac{f^{(n+1)}(x_0+\theta(x-x_0))}{(n+1)!}(x-x_0)^{n+1}, \theta\in(0,1)$
KL divergence: $D_{\text{KL}}(P||Q) = -\sum_i P(i)\log\frac{Q(i)}{P(i)} = \sum_x P(x)\log\left(\frac{P(x)}{Q(x)}\right)$
Log Likelihood: $\log\Pi_n^N p(t_n|x_n) = \log\Pi_n^N p(t_n = 1|x_n)^{t_n}p(t_n = 0|x_n)^{1-t_n} = \sum_n^N t_n\log p(t_n|x_n) + (1-t_n)\log(1-p(t_n|x_n))$
put minus in front to get cross-entropy loss.
Probability Names: $P(Y|X)\to$ likelihood, $P(X|Y)\to$ posterior, $P(X)\to$ prior, $P(Y)\to$ evidence. Posterior = (likelihood * prior)/evidence
Set Theory Compact = closed and bounded
$f\in\mathcal{C}^{n+1}([a,b],\mathbb{R})$: diff. $n+1$ times and is a mapping from $[a,b]$ to $\mathbb{R}$
Newton's Method: $\theta(t+1) = \theta(t) - \underbrace{(\nabla^2\mathcal{R})^{-1}}_{\text{inverse Hessian}}\nabla\mathcal{R}\big|_{\theta=\theta(t)}$
Sylvester's criterion: necessary and sufficient criterion to determine whether a Hermitian matrix is positive-definite. All upper left corner matrices have to have a positive determinant.
Weierstrass theorem: If f is a given continuous function for $a\leq x\leq b$ and if ϵ is an arbitrary positive quantity, it is possible to construct an approximating polynomial $P(x)$ such that: $|f(x)-P(x)|\leq\epsilon$, $a\leq x\leq b$

Misc. Deep Learning

Loss Functions

L1-norm loss: least absolute deviations (LAD), least absolute errors (LAE)
$L = \sum_{i=1}^n|y_i - f(x_i)|$
L2-norm loss: least-squared errors (LSE)
$L = \sum_{i=1}^n\|y_i - f(x_i)\|^2$
L2-regularized:
$L(x;\theta) = \sum_{i=1}^n\|y_i - f(x_i;\theta)\|^2 + \lambda\sum_{ij}\theta_{ji}^2$
Mean Squared Error: MSE
$L = \sum_{i=1}^n\|y_i - f(x_i)\|^2/n$
Cross-Entropy Loss: log loss
$L = -\log\Pi_n^N p(t_n|x_n) = -\log\Pi_n^N p(t_n = 1|x_n)^{t_n}p(t_n = 0|x_n)^{1-t_n} = -\left(\sum_n^N t_n\log p(t_n|x_n) + (1-t_n)\log(1-p(t_n|x_n))\right)$