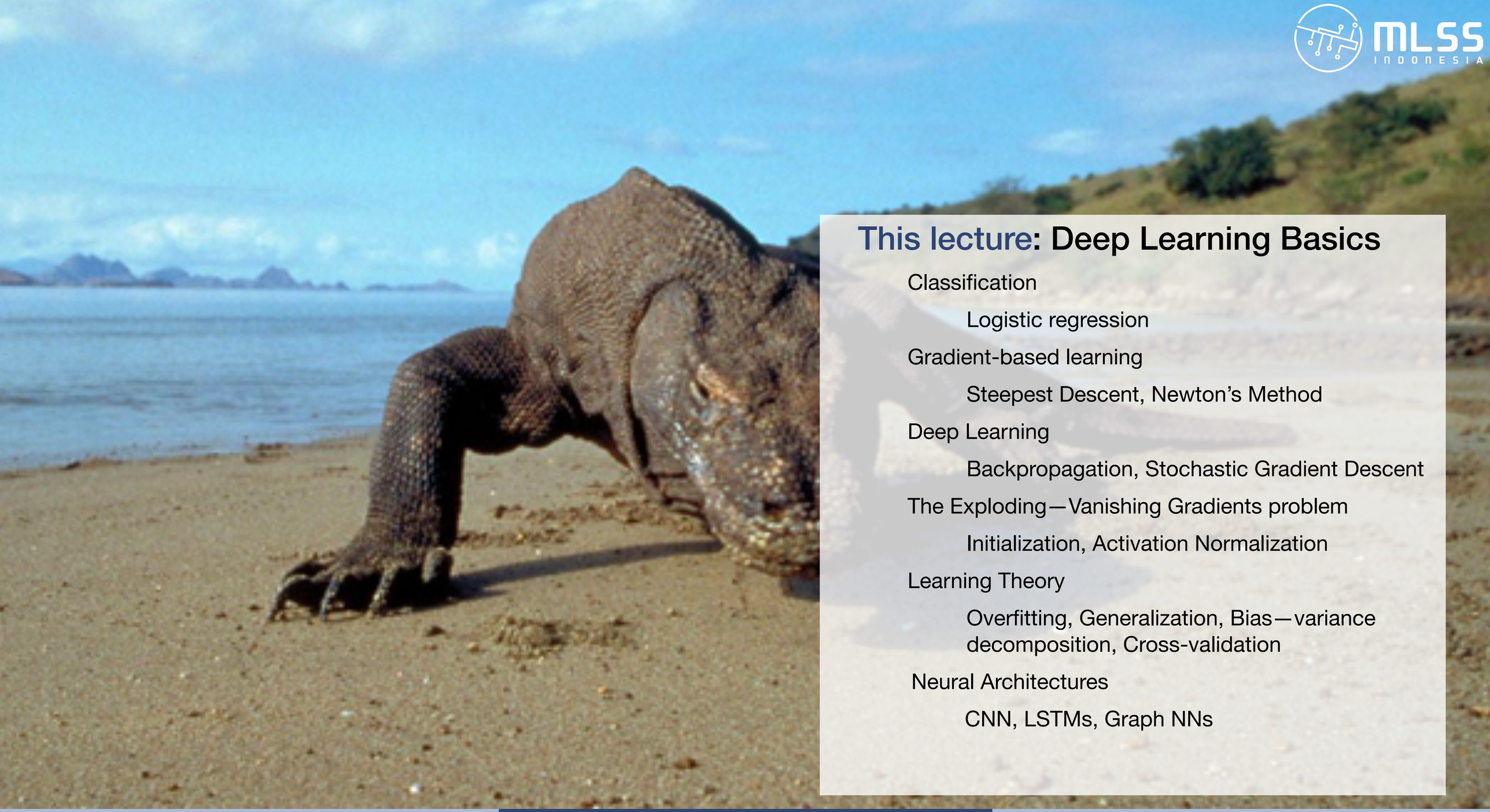
Deep Learning Basics: Lecture 2
Machine Learning Summer School, Indonesia 2020

Daniel Worrall

Komodo dragon



This lecture: Deep Learning Basics

- Classification
  - Logistic regression
- Gradient-based learning
  - Steepest Descent, Newton's Method
- Deep Learning
  - Backpropagation, Stochastic Gradient Descent
- The Exploding—Vanishing Gradients problem
  - Initialization, Activation Normalization
- Learning Theory
  - Overfitting, Generalization, Bias—variance decomposition, Cross-validation
- Neural Architectures
  - CNN, LSTMs, Graph NNs

Logistic regression

Pulau Komodo

Logistic Regression

Logistic regression[1] is a classification technique. It is a *discriminative model*.

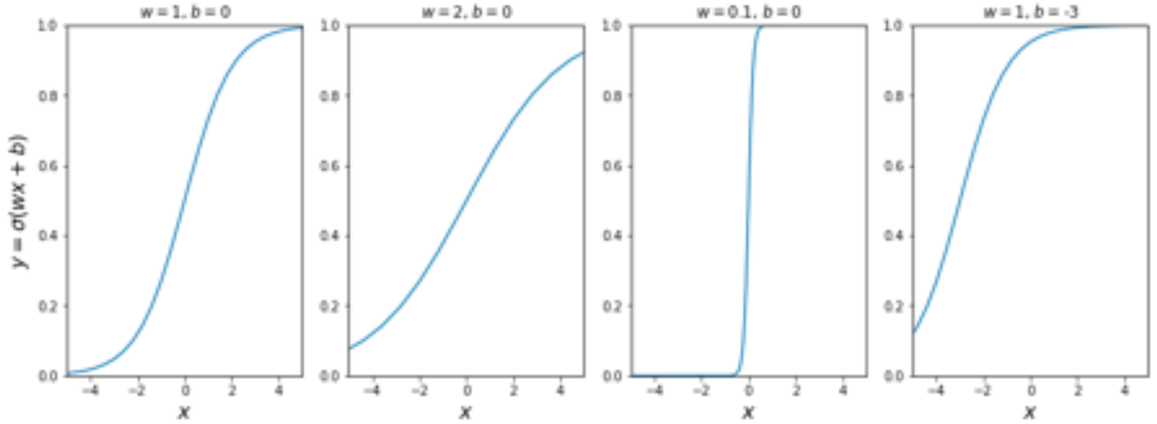
Binary logistic regression

Input: $\mathbf{x} \in \mathbb{R}^D$ **Output:** $y \in \{0, 1\}$

Training data: $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$

Model

$$p(y|\mathbf{x}, \mathbf{w}) = \text{Bernoulli}(y|\pi(\mathbf{x}))$$
$$= \pi^y(\mathbf{x})(1 - \pi(\mathbf{x}))^{1-y}$$

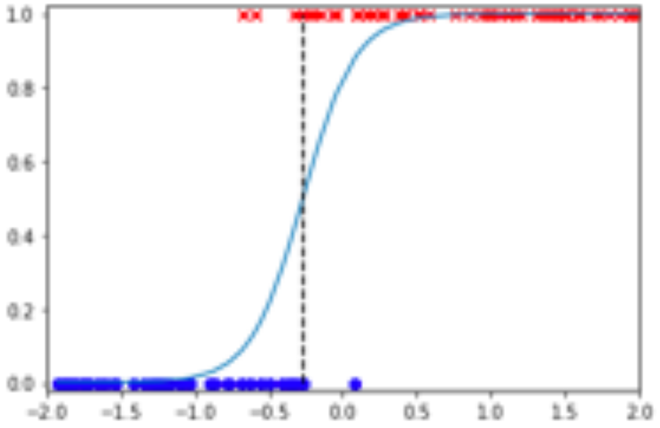


We classify an input with label 1 if $p(y = 1|\mathbf{x}, \mathbf{w}) > 0.5$

where $\pi(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$

Sigmoid activation function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Logistic regression not in conjugate exponential family: **no conjugate prior**.

Methods exist to approximate posterior over the parameters:

Variational Inference, Monte Carlo sampling etc.,

We choose to just do maximum likelihood

Note, we use the following feature transform to include a bias:

$$\mathbf{w}^\top \mathbf{x} = \begin{bmatrix} w & b \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

[1] It should really be called logistic classification

Logistic Regression: Max. Likelihood

The negative log likelihood is

$$\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N} \log p(y_n|\mathbf{x}_n, \mathbf{w})$$

Cross-entropy

$$= -\sum_{n=1}^{N} y_n \log \sigma(\mathbf{w}^\top \mathbf{x}_n) + (1-y_n)\log(1-\sigma(\mathbf{w}^\top \mathbf{x}_n))$$

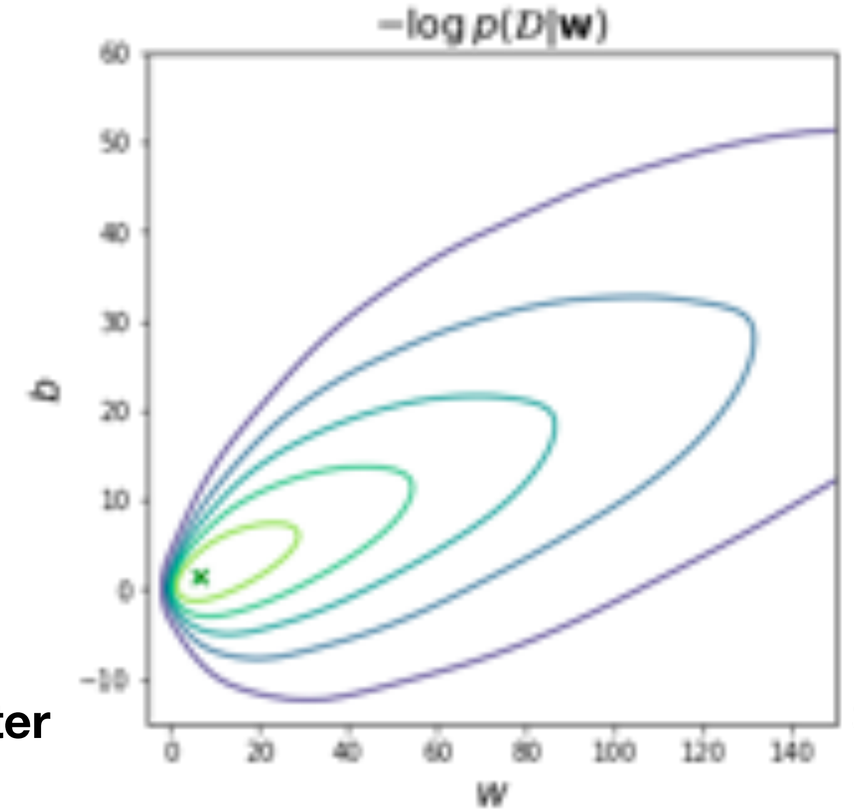
The gradients

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = -\sum_{n=1}^{N} y_n \frac{\partial}{\partial \mathbf{w}} \log \sigma(\mathbf{w}^\top \mathbf{x}_n) + (1-y_n)\frac{\partial}{\partial \mathbf{w}} \log(1-\sigma(\mathbf{w}^\top \mathbf{x}_n))$$

$$= -\sum_{n=1}^{N} y_n(1-\sigma(\mathbf{w}^\top \mathbf{x}_n))\mathbf{x}_n - (1-y_n)\sigma(\mathbf{w}^\top \mathbf{x}_n)\mathbf{x}_n$$

$$= -\sum_{n=1}^{N} (y_n - \sigma(\mathbf{w}^\top \mathbf{x}_n))\mathbf{x}_n$$

Prediction error



Flatter **Sharper**

$$\frac{\partial \sigma}{\partial z} = \frac{\partial}{\partial z}(1+e^{-z})^{-1} = \frac{1}{1+e^{-z}}\frac{e^{-z}}{1+e^{-z}} = \sigma(z)(1-\sigma(z))$$

$$\frac{\partial \log \sigma}{\partial z} = \frac{1}{\sigma(z)}\frac{\partial \sigma}{\partial z} = \frac{\sigma(z)}{\sigma(z)}(1-\sigma(z)) = 1-\sigma(z)$$

$$\frac{\partial \log(1-\sigma(z))}{\partial z} = \frac{1}{1-\sigma(z)}\frac{\partial(1-\sigma(z))}{\partial z} = -\frac{1-\sigma(z)}{1-\sigma(z)}\sigma(z) = -\sigma(z)$$

Matrix calculus

We use *numerator layout convention*, also called *Jacobian convention*.

Column vector

$$\frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} \\ \vdots \\ \frac{\partial y_M}{\partial x_1} \end{bmatrix}$$

Row vector

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_N} \end{bmatrix}$$

Matrix

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_M}{\partial x_1} & \cdots & \frac{\partial y_M}{\partial x_N} \end{bmatrix}$$

Chain rule

$$\mathbf{y} = \boldsymbol{\phi}(\boldsymbol{\psi}(\mathbf{x}))$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{\psi}} \frac{\partial \boldsymbol{\psi}}{\partial \mathbf{x}}$$

e.g.

$$\frac{\partial \mathbf{W}\mathbf{x}}{\partial \mathbf{x}} = \mathbf{W} \qquad \frac{\partial y_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_k W_{ik} x_k = \sum_k W_{ik} \frac{\partial x_k}{\partial x_j} = \sum_k W_{ik} \delta_{kj} = W_{ij}$$

$$\frac{\partial \mathbf{x}^\top \mathbf{W}}{\partial \mathbf{x}} = \mathbf{W}^\top \qquad \frac{\partial y_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_k x_k W_{ki} = \sum_k \frac{\partial x_k}{\partial x_j} W_{ki} = \sum_k \delta_{kj} W_{ki} = W_{ji}$$

More examples in https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

Multiclass classification

Multiclass logistic

Input: $\mathbf{x} \in \mathbb{R}^D$ **Output:** $y \in \{0, 1, ..., C-1\}$

Training data: $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$

Model $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \text{Categorical}(\mathbf{y}|\boldsymbol{\pi}(\mathbf{x}))$

One-hot vector $\mathbf{y} = \underbrace{[0, 0, .., 1, ..., 0]^\top}_{C \text{ classes}}$

Build probability vectors with a multiclass generalization of the sigmoid

$\boldsymbol{\pi}(\mathbf{x}) = \text{softmax}(\mathbf{W}\mathbf{x})$

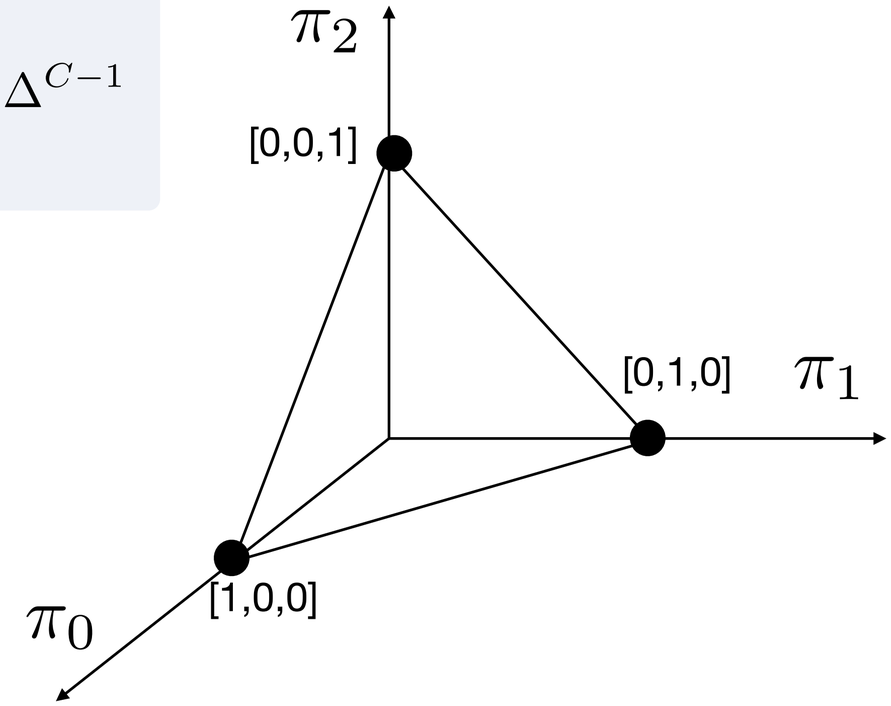
$$\text{softmax}(\mathbf{z})_c = \frac{e^{z_c}}{\sum_{k=0}^{C-1} e^{z_k}}$$

Multiclass classification: *N*D generalization of the Bernoulli is the Categorical distribution

$$\text{Categorical}(y|\boldsymbol{\pi}) = \prod_{c=0}^{C-1} \pi_c^{y_c}$$

$$\sum_{c=0}^{C-1} \pi_c = 1 \iff \boldsymbol{\pi} \in \Delta^{C-1}$$

Probability simplex



Multiclass classification

The negative log-likelihood is very simple

$$
\begin{aligned}
\mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N} \log p(\mathbf{y}_n | \mathbf{x}_n, \mathbf{w}) \\
&= -\sum_{n=1}^{N} \log \prod_{c=0}^{C-1} \pi_c(\mathbf{x}_n)^{y_{n,c}} \\
&= -\sum_{n=1}^{N} \sum_{c=0}^{C-1} y_{n,c} \log \pi_c(\mathbf{x}_n)
\end{aligned}
$$

The derivative for a single point has a satisfying result

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial z_i} &= -\frac{\partial}{\partial z_i} \sum_{c=0}^{C-1} y_c \log \operatorname{softmax}(\mathbf{z})_c \\
&= -\frac{\partial}{\partial z_i} \sum_{c=0}^{C-1} y_c \log \frac{e^{z_c}}{\sum_k e^{z_k}} \\
&= -\frac{\partial}{\partial z_i} \sum_{c=0}^{C-1} y_c z_c - y_{n,c} \log \sum_k e^{z_k} \\
&= -\frac{\partial}{\partial z_i} \left(\sum_{c=0}^{C-1} y_c z_c \right) + \log \sum_k e^{z_k} \\
&= -y_i + \frac{e^{z_i}}{\sum_k e^{z_k}} \\
&= -(y_i - \operatorname{softmax}(\mathbf{z})_i)
\end{aligned}
$$

Prediction error

Gradient-based learning

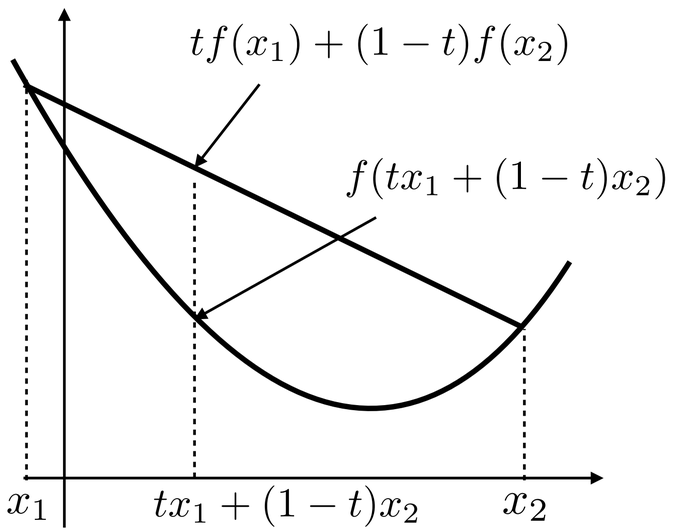
Jatiluwih Rice Terraces, Bali

Gradient descent

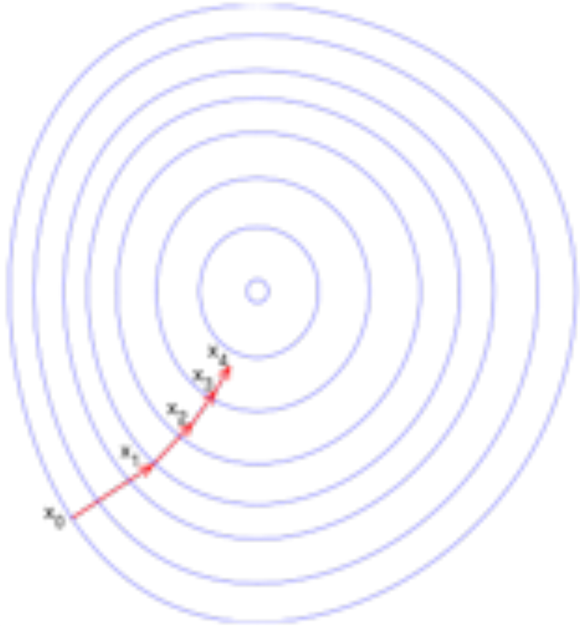
The following expression is *intractable*!

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = -\sum_{n=1}^{N}(y_n - \sigma(\mathbf{w}^\top \mathbf{x}_n))\mathbf{x}_n = \mathbf{0}$$

Gradient descent is a *numerical optimization* scheme to approximate the minimizer for *convex* loss functions



$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$



Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \mathbf{g}_t \qquad \mathbf{g}_t = \left.\frac{\partial \mathcal{L}}{\partial \mathbf{w}}\right|_{\mathbf{w}_t}$$

for small enough $\lambda_t \geq 0$

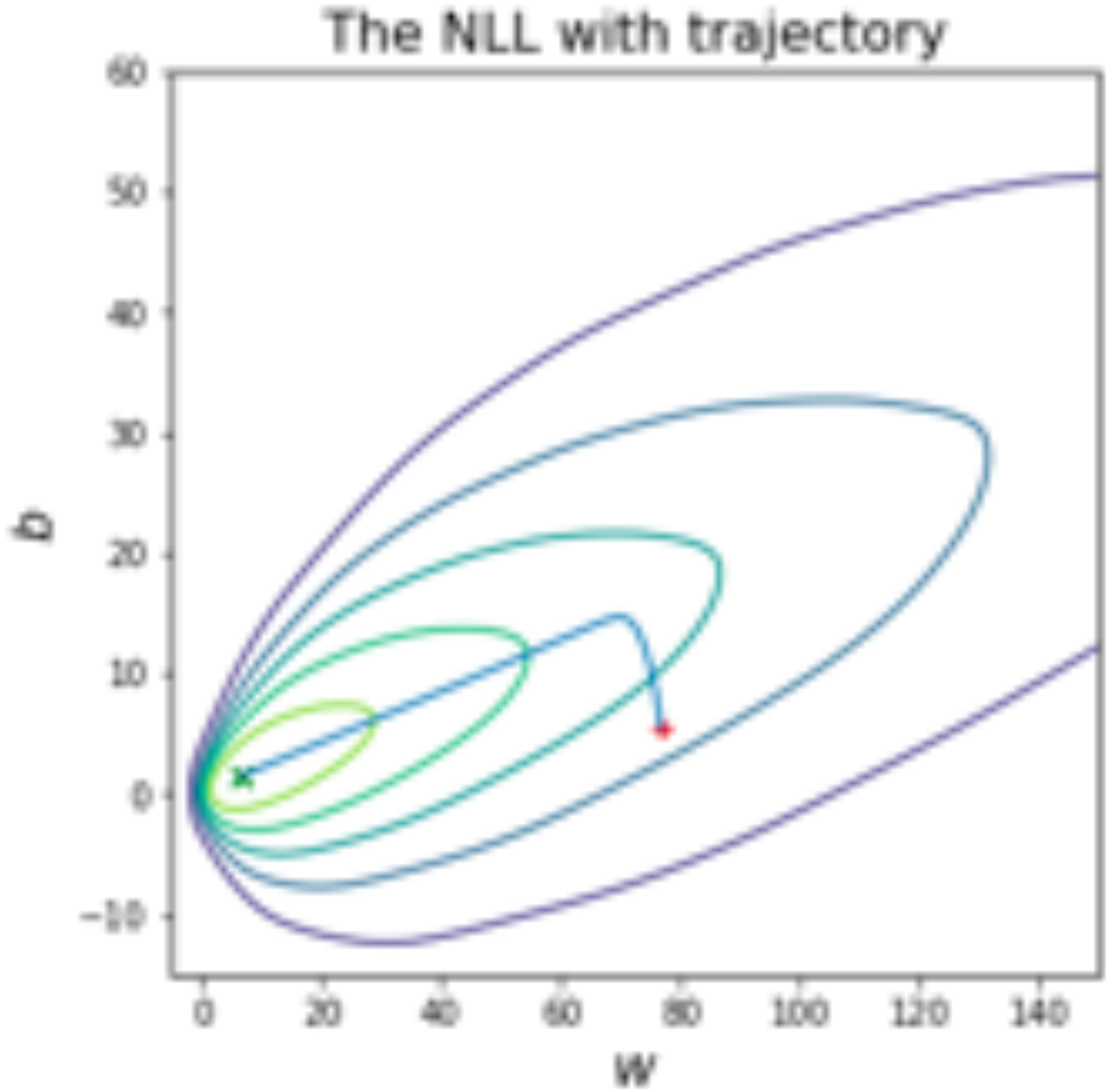
Non-divergence $\mathcal{L}(\mathbf{w}_{t+1}) \leq \mathcal{L}(\mathbf{w}_t)$

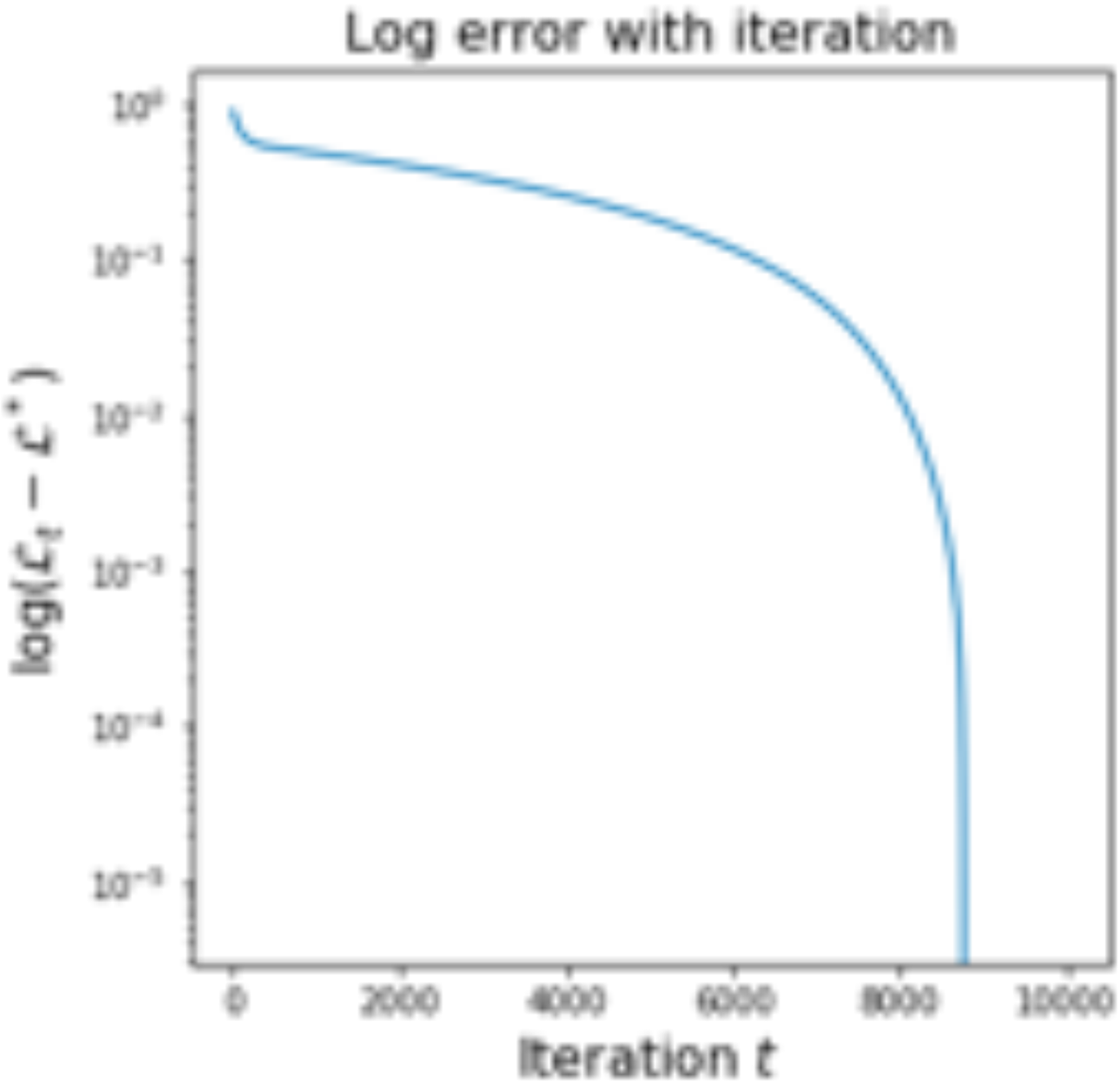
Convergence rate: Linear (m=1)

$$\|\mathbf{w}_t - \mathbf{w}^*\| \leq \left(\frac{1}{t}\right)^m$$

1. Initialize $\mathbf{w}_0$
2. Compute update $\Delta \mathbf{w}_t = -\lambda_t \mathbf{g}_t$
3. Merge update $\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta \mathbf{w}_t$
4. Terminate?
5. Goto 2 and set $t \leftarrow t + 1$

Logistic Regression: Max. Likelihood





Gradient descent: Newton's methods

Gradient descent is a 1st order method. A faster optimization scheme is *Newton's method*.

2nd order Taylor

$$\mathcal{L}(\mathbf{w}+\Delta\mathbf{w}) \simeq \mathcal{L}(\mathbf{w}) + \Delta\mathbf{w}^\top \mathbf{g}_t + \frac{1}{2}\Delta\mathbf{w}^\top \mathbf{H}_t \Delta\mathbf{w}$$

Derivative wrt update $\Delta\mathbf{w}$

$$\frac{\partial}{\partial \Delta\mathbf{w}}\mathcal{L}(\mathbf{w}+\Delta\mathbf{w}) \simeq \mathbf{g}_t + \mathbf{H}_t\Delta\mathbf{w} = \mathbf{0}$$

Optimal update

$$\implies \Delta\mathbf{w} = -\mathbf{H}_t^{-1}\mathbf{g}_t$$

Hessian

$$\mathbf{H}_t = \frac{\partial^2 \mathcal{L}}{\partial \mathbf{w}\partial \mathbf{w}^\top}\bigg|_{\mathbf{w}_t}$$

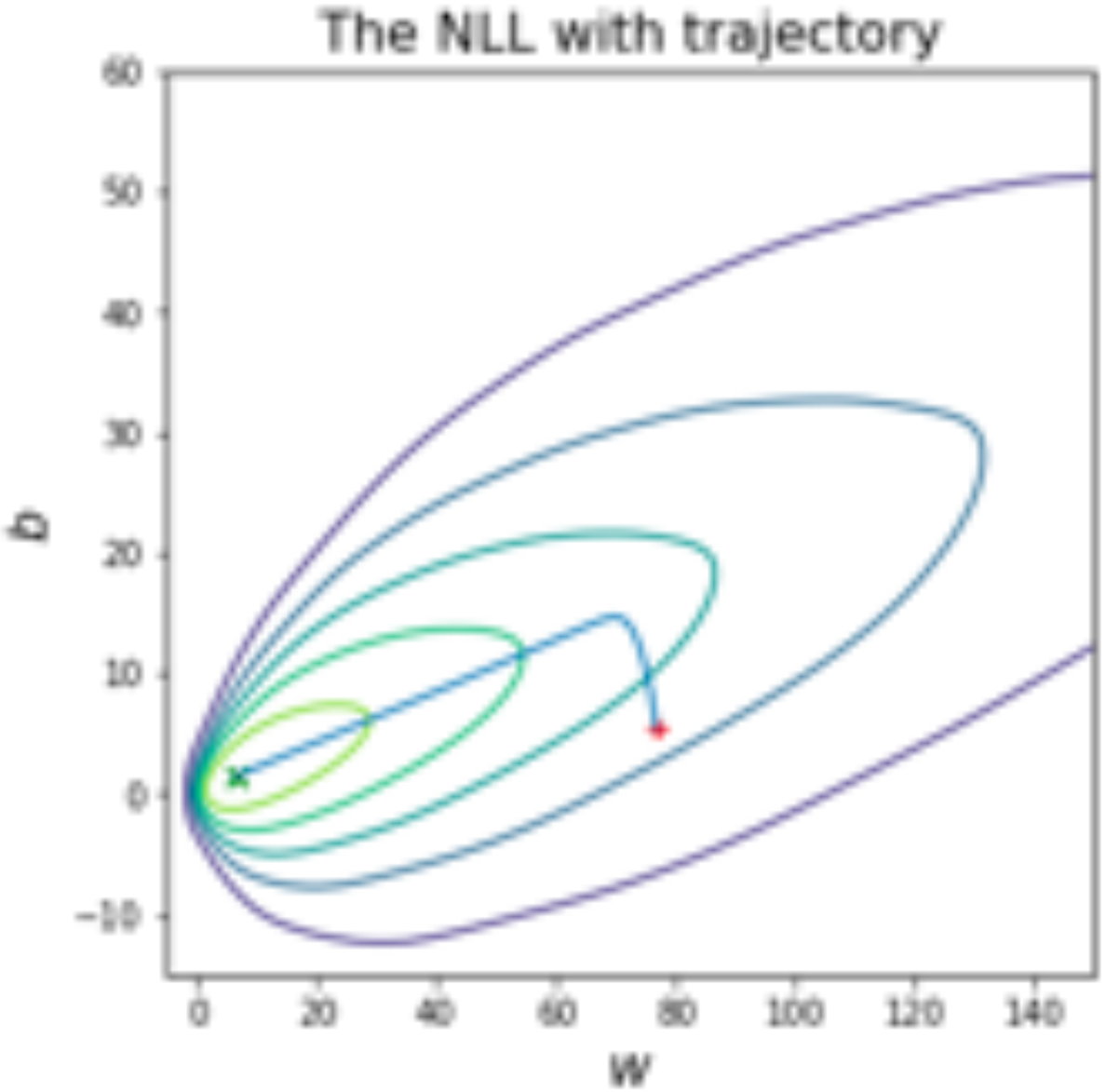
We see that the 1st order scheme corresponds to approximating $\mathbf{H}_t = \mathbf{I}$

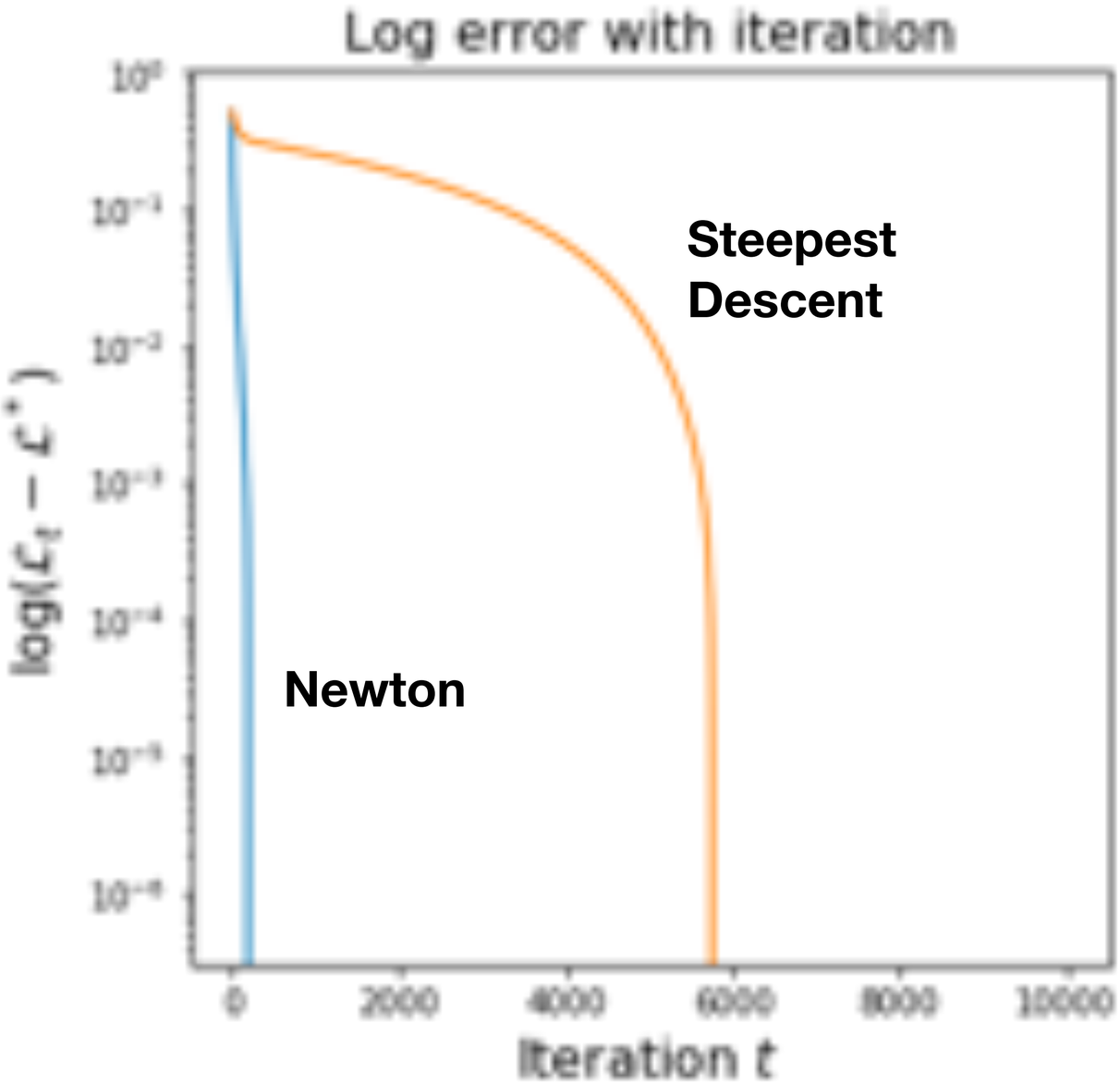
Newton's method offers quadratic convergence $\mathcal{O}\left(1/T^2\right)$, but it also requires the inversion of a $D \times D$ matrix in $\mathcal{O}(D^3)$ time

Logistic regression

$$\begin{aligned}
\mathbf{H}_t &= \frac{\partial}{\partial \mathbf{w}^\top}\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}^\top}\sum_{n=1}^{N}(y_n - \sigma(\mathbf{w}^\top \mathbf{x}_n))\mathbf{x}_n \\
&= -\sum_{n=1}^{N}\frac{\partial}{\partial \mathbf{w}^\top}\sigma(\mathbf{w}^\top \mathbf{x}_n)\mathbf{x}_n \\
&= -\sum_{n=1}^{N}\sigma(\mathbf{w}^\top \mathbf{x}_n)\sigma(1-\mathbf{w}^\top \mathbf{x}_n)\mathbf{x}_n\mathbf{x}_n^\top
\end{aligned}$$

Logistic Regression: Max. Likelihood





Decision boundaries

Decision boundaries are surfaces where a classification switches classes. For the logistic regression the decision boundaries are *hyperplanes*.

$$p(y|\mathbf{x}, \mathbf{w}) = \text{Bernoulli}(y|\pi(\mathbf{x})) = \pi^y(\mathbf{x})(1-\pi(\mathbf{x}))^{1-y}$$

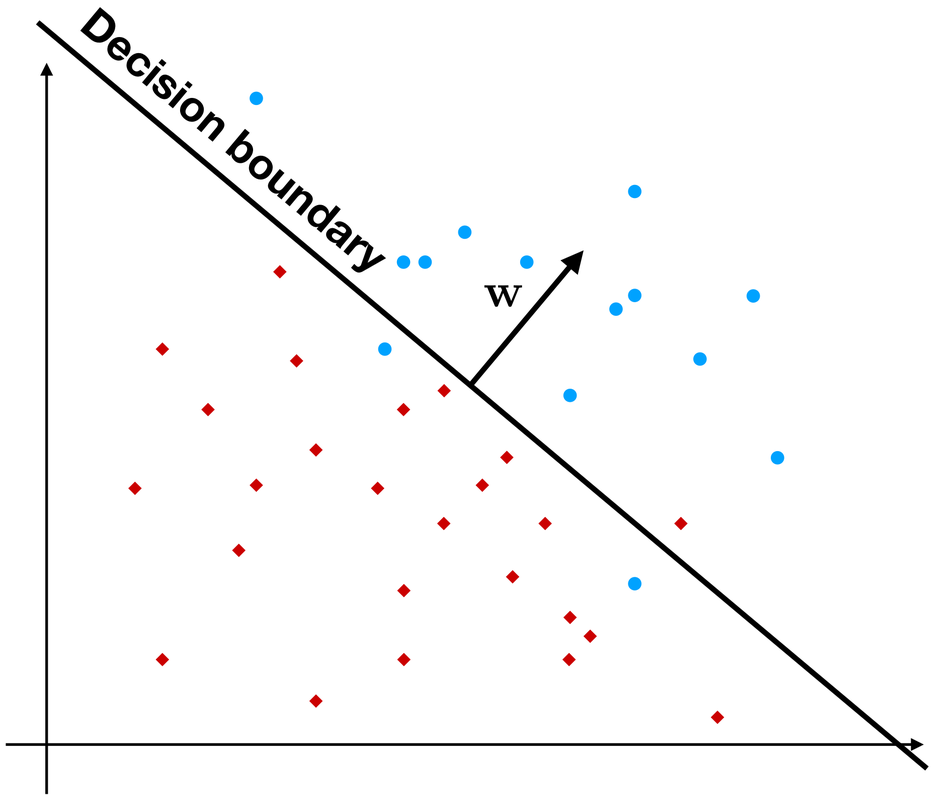
For positive classification

$$\sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}} > 0.5$$

$$e^{-\mathbf{w}^\top \mathbf{x}} \leq 1 \implies \mathbf{w}^\top \mathbf{x} > 0$$

$\mathbf{w}$ is a normal vector to a binary decision hyperplane

How do we get more flexible decision boundaries?



Shallow features: The multilayer perceptron

Idea: use a learnable feature transform

$$\boldsymbol{\phi}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Bias

Weight matrix

Nonlinearity/activation function

Universal Approximation Theorem

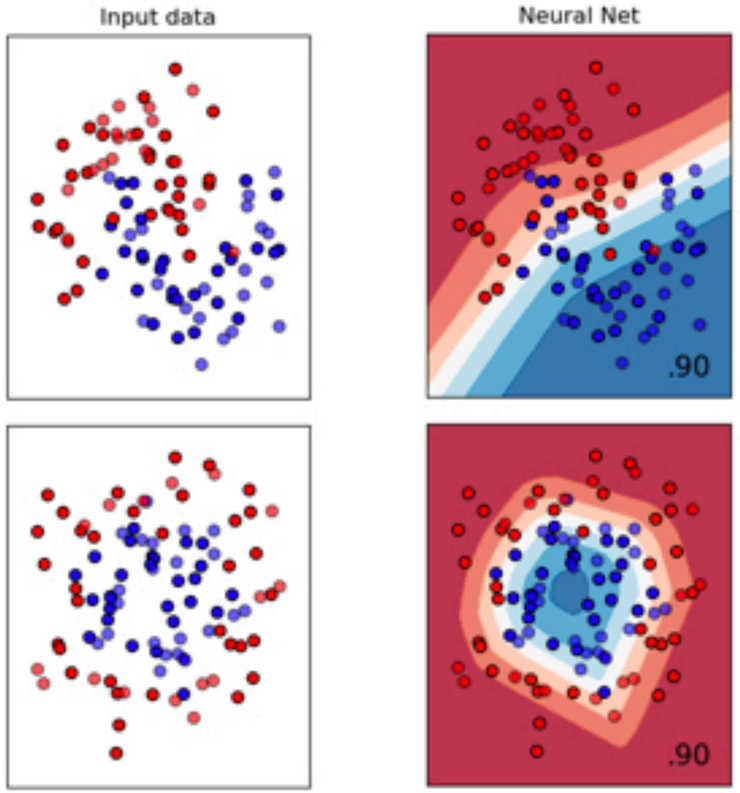
For non-constant, bounded, continuous, element-wise σ, and large enough internal dimension, $f(\mathbf{x})$ can approximate any real-valued, continuous function on the hypercube to arbitrary accuracy:

$$f(\mathbf{x}) = \mathbf{w}^\top \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$p(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \text{Categorical}(\mathbf{y}|\boldsymbol{\pi}(\mathbf{x}))$$

$$\boldsymbol{\pi}(\mathbf{x}) = \text{softmax}(\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}))$$

Can learn any function, and any decision boundary!



https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Deep Learning

Corals, Maluku

Deep features

In the last decades, however, researchers have found shallow features to generalize poorly. Empirically, stacking learnable feature transforms performs better:

$$f(\mathbf{x}) = \boldsymbol{\phi}_L(\boldsymbol{\phi}_{L-2}(\cdots \boldsymbol{\phi}_1(\mathbf{x}))) = \boldsymbol{\phi}_L \circ \boldsymbol{\phi}_{L-1} \circ \cdots \circ \boldsymbol{\phi}_1(\mathbf{x})$$

Function composition

Just why this is the case, we do not fully understand yet.

Training

Gradients of loss wrt $\mathbf{W}_\ell$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{W}_\ell} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}_L} \frac{\partial \boldsymbol{\phi}_L}{\partial \mathbf{W}_\ell} \\
&= \frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}_L} \frac{\partial \boldsymbol{\phi}_L}{\partial \boldsymbol{\phi}_{L-1}} \frac{\partial \boldsymbol{\phi}_{L-1}}{\partial \mathbf{W}_\ell} \\
&= \frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}_L} \frac{\partial \boldsymbol{\phi}_L}{\partial \boldsymbol{\phi}_{L-1}} \cdots \frac{\partial \boldsymbol{\phi}_\ell}{\partial \mathbf{W}_\ell}
\end{aligned}$$

$$\left[\frac{\partial \boldsymbol{\phi}_L}{\partial \boldsymbol{\phi}_{L-1}}\right] = D \times D$$

$$\text{Cost}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}_\ell}\right) = \mathcal{O}((L-\ell)D^3)$$

Cost of all gradients: $\mathcal{O}(L^2 D^3)$

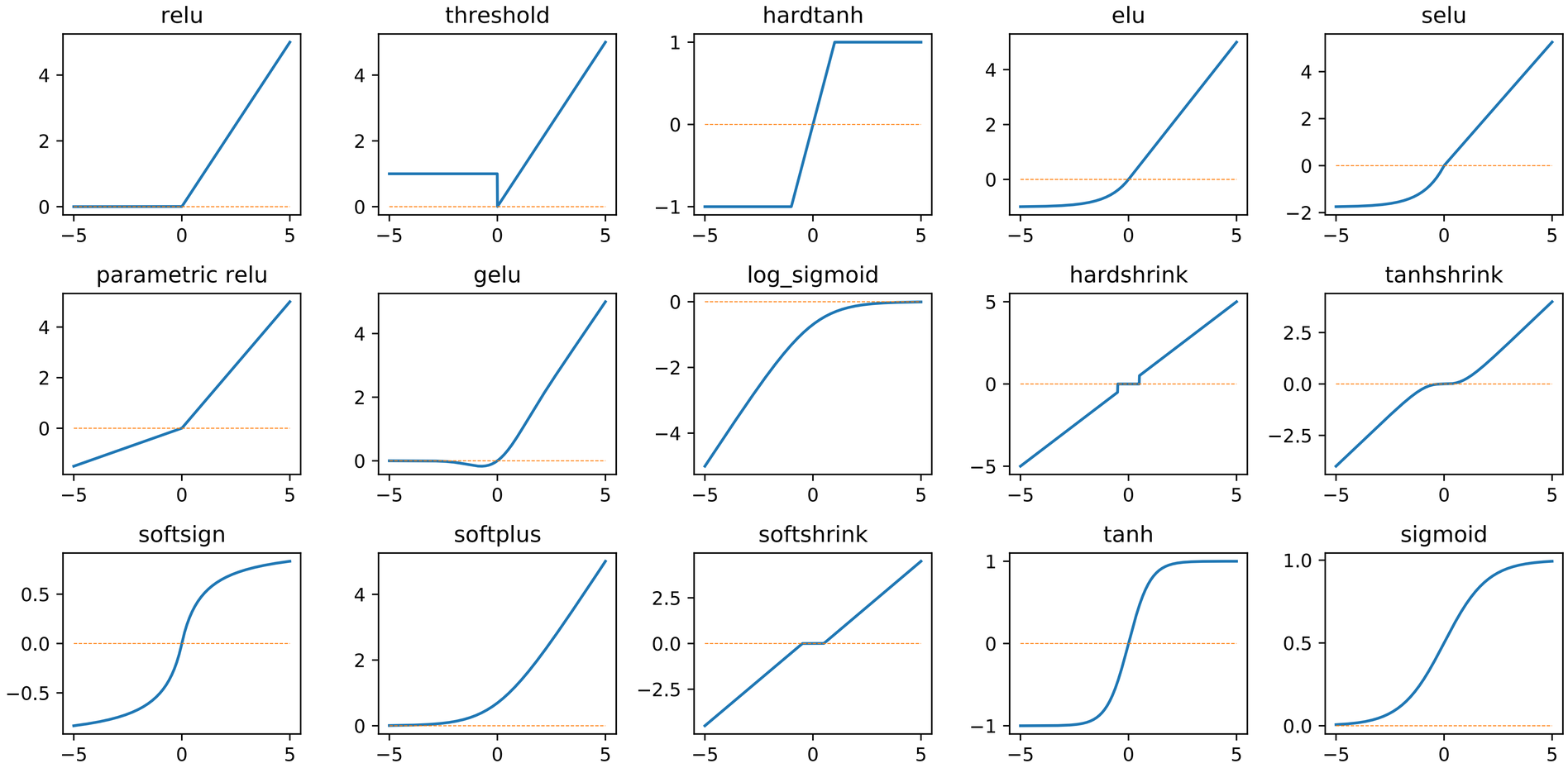
Nonlinearities/Activation functions

Choosing a nonlinearity is hard..

- Most people use the ReLU = max(x,0)
- Best just to try and see

Consideration

Slope saturation?
Discontinuities?
Symmetries?



The Backpropagation Algorithm

Recursive gradient computation:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{\ell+1}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}_{\ell+1}} \frac{\partial \boldsymbol{\phi}_{\ell+1}}{\partial \mathbf{W}_{\ell+1}} \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{\ell}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}_{\ell+1}} \frac{\partial \boldsymbol{\phi}_{\ell+1}}{\partial \boldsymbol{\phi}_{\ell}} \frac{\partial \boldsymbol{\phi}_{\ell}}{\partial \mathbf{W}_{\ell}}$$

Notice that both gradients **reuse** the same Jacobian evaluations: *overlapping subproblems*.

Define reverse-mode derivative: $\boldsymbol{\Delta}_\ell = \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}_\ell}$

init: $\boldsymbol{\Delta}_L = \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}_L}$

for $\ell = L, \ldots, 1$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_\ell} = \boldsymbol{\Delta}_\ell \frac{\partial \boldsymbol{\phi}_\ell}{\partial \mathbf{W}_\ell}$$

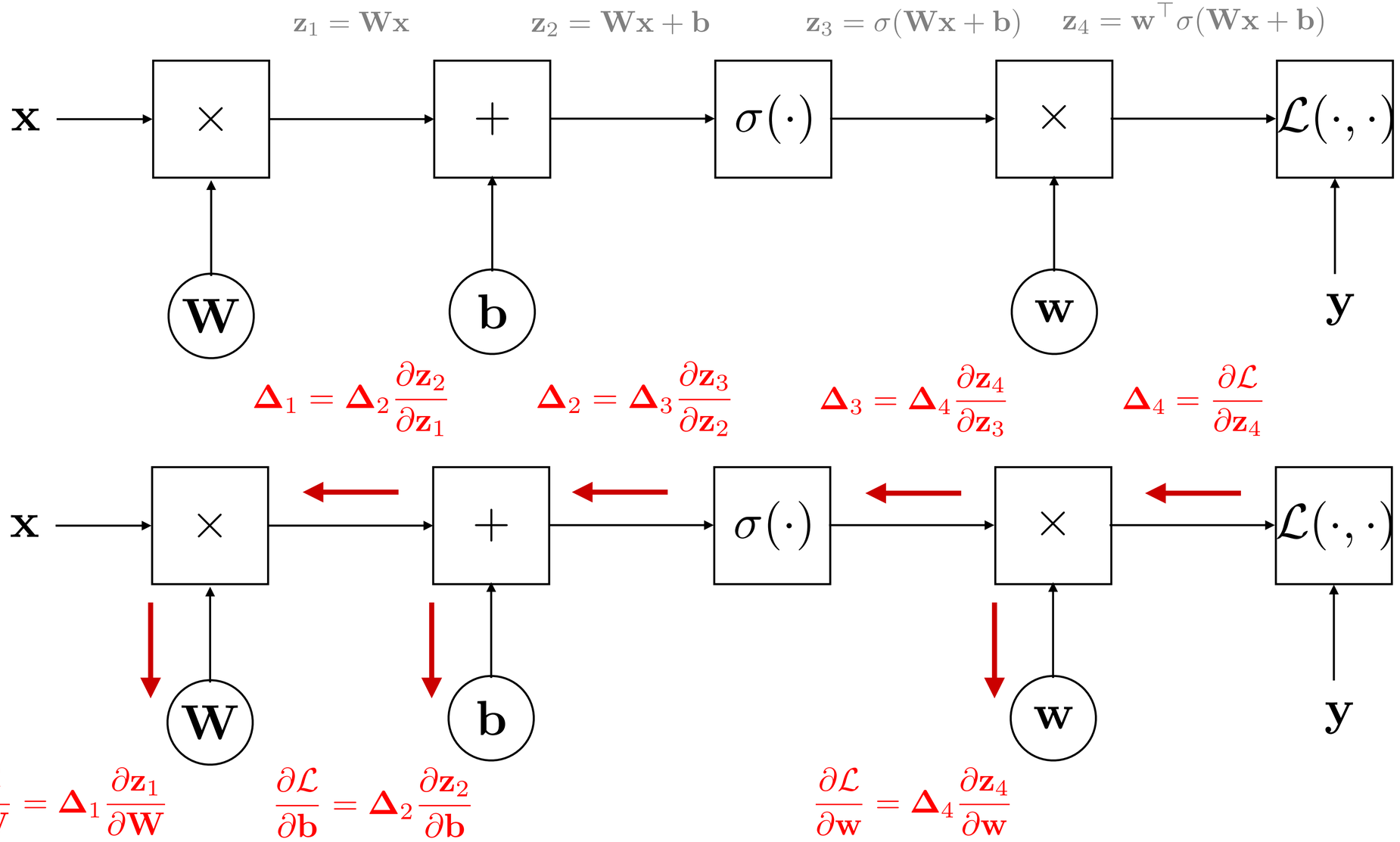
$$\boldsymbol{\Delta}_{\ell-1} = \boldsymbol{\Delta}_\ell \frac{\partial \boldsymbol{\phi}_\ell}{\partial \boldsymbol{\phi}_{\ell-1}}$$

Complexity: $\mathcal{O}(LD^3)$

Need to store activations in forward pass, before applying backward

Backpropagation

We can represent the how a feature is formed using a *computation graph*:



Stochastic Gradient Descent

Typical loss functions are sums

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}} \sum_{n=1}^{N} \mathcal{L}_n(\mathbf{w}) = \sum_{n=1}^{N} \frac{\partial \mathcal{L}_n}{\partial \mathbf{w}}$$

where $\mathcal{L}_n(\mathbf{w}) = -\log p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{w})$

	Single gradient	Single update
Time complexity	$\mathcal{O}(LD^3)$	$\mathcal{O}(NLD^3)$
Space complexity	$\mathcal{O}(LD^2)$	$\mathcal{O}(NLD^2)$

Stochastic Gradients: Estimate gradient on random mini-batch

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} \simeq \tilde{\mathbf{g}} = \frac{N}{M} \sum_{m=1}^{M} \mathbf{g}_m \qquad \text{where } \mathbf{g} \sim \text{Uniform}\left(\left\{\frac{\partial \mathcal{L}_1}{\partial \mathbf{w}}, \frac{\partial \mathcal{L}_2}{\partial \mathbf{w}}, ..., \frac{\partial \mathcal{L}_N}{\partial \mathbf{w}}\right\}\right)$$

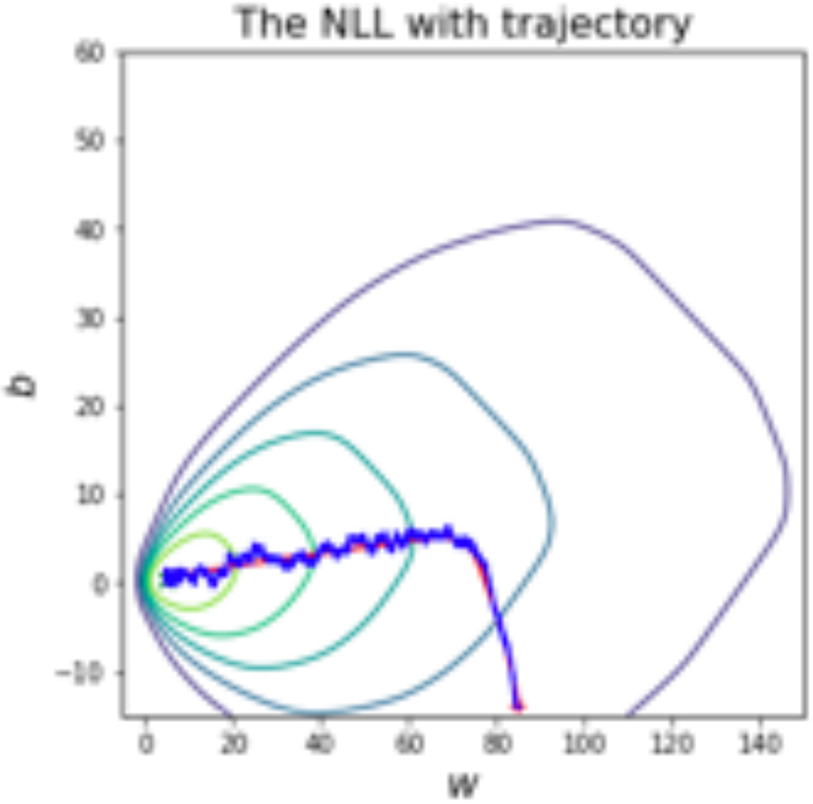
This is an unbiased estimate of the true gradient.

Robbins—Monro conditions: $O(1/T)$ converge if

$$\sum_{t=1}^{T} \lambda_t = \infty, \quad \sum_{t=1}^{T} \lambda_t^2 < \infty$$

Converge from any initial point

Limit noise from stochasticity



SGD Variants

Noise reduction achieved with smoothing

Exponential smoother

Momentum: gradient averaging

$$\boldsymbol{\mu}_t = \alpha_t \boldsymbol{\mu}_{t-1} + (1 - \alpha_t)\tilde{\mathbf{g}}_t$$
$$\mathbf{w} = \mathbf{w}_{t-1} - \lambda_t \boldsymbol{\mu}_t$$

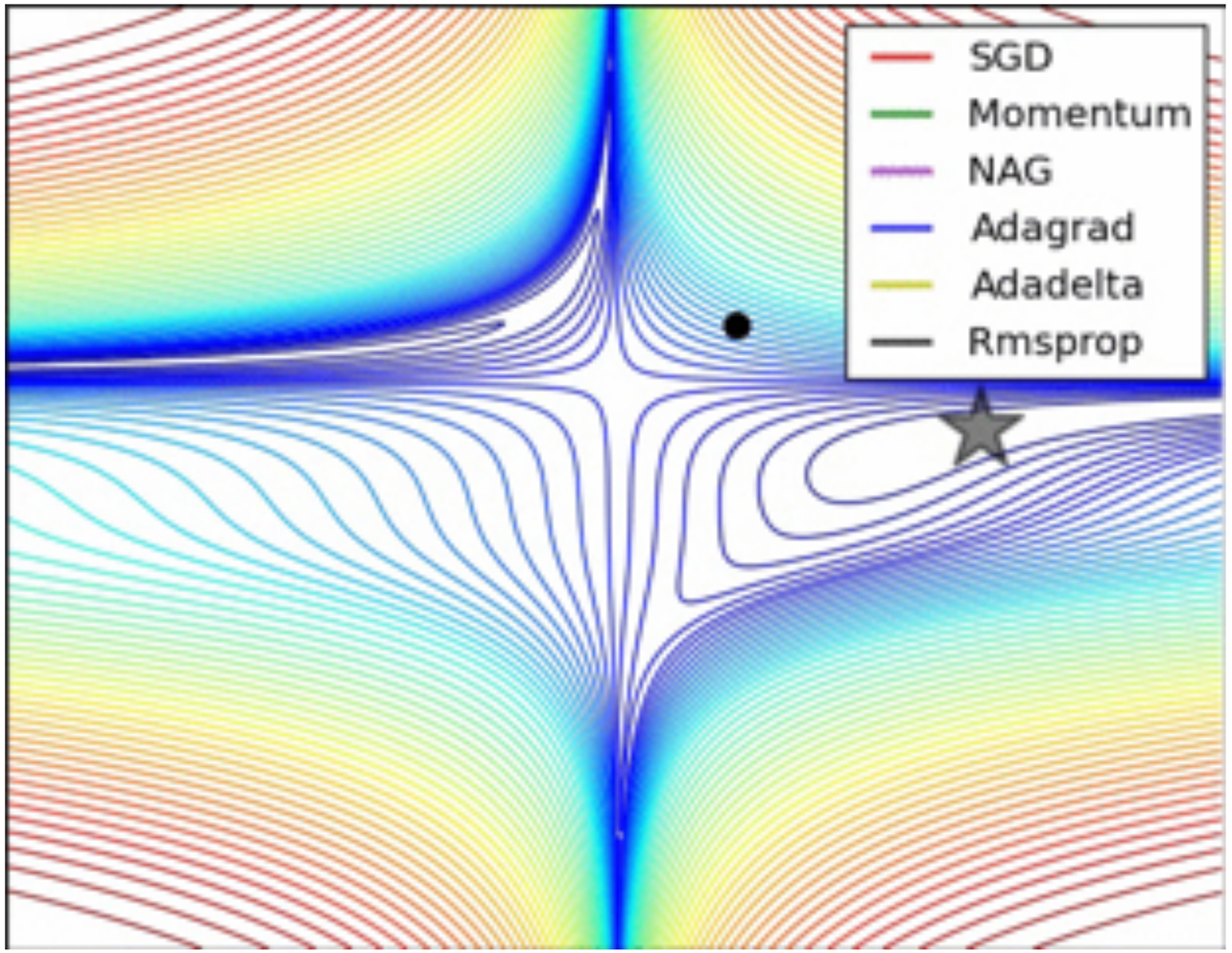
RMSProp: learning rate normalization

$$\boldsymbol{\nu}_t = \alpha_t \boldsymbol{\nu}_{t-1} + (1 - \alpha_t)\tilde{\mathbf{g}}_t^2$$
$$\mathbf{w} = \mathbf{w}_{t-1} - \frac{\lambda_t \tilde{\mathbf{g}}_t}{\sqrt{\boldsymbol{\nu}_t}}$$

element-wise

Large noise, small learning rate

$$\left.\begin{aligned} \boldsymbol{\nu}_t &\simeq \boldsymbol{\mu}_t^2 + \boldsymbol{\sigma}_t^2 \\ \mathbb{E}[\tilde{\mathbf{g}}_t] &\simeq \boldsymbol{\mu}_t \end{aligned}\right\} \Rightarrow \frac{\tilde{\mathbf{g}}_t}{\sqrt{\nu}_t} \simeq \frac{\boldsymbol{\mu}_t}{\sqrt{\boldsymbol{\mu}_t^2 + \boldsymbol{\sigma}_t^2}} \leq 1$$



https://ruder.io/content/images/2016/09/contours_evaluation_optimizers.gif

SGD Variants

Noise reduction achieved with smoothing

Exponential smoother

Momentum: gradient averaging

$$\boldsymbol{\mu}_t = \alpha_t \boldsymbol{\mu}_{t-1} + (1-\alpha_t)\tilde{\mathbf{g}}_t$$
$$\mathbf{w} = \mathbf{w}_{t-1} - \lambda_t \boldsymbol{\mu}_t$$

RMSProp: learning rate normalization

$$\boldsymbol{\nu}_t = \alpha_t \boldsymbol{\nu}_{t-1} + (1-\alpha_t)\tilde{\mathbf{g}}_t^2$$
$$\mathbf{w} = \mathbf{w}_{t-1} - \frac{\lambda_t \tilde{\mathbf{g}}_t}{\sqrt{\boldsymbol{\nu}_t}}$$

element-wise

ADAM

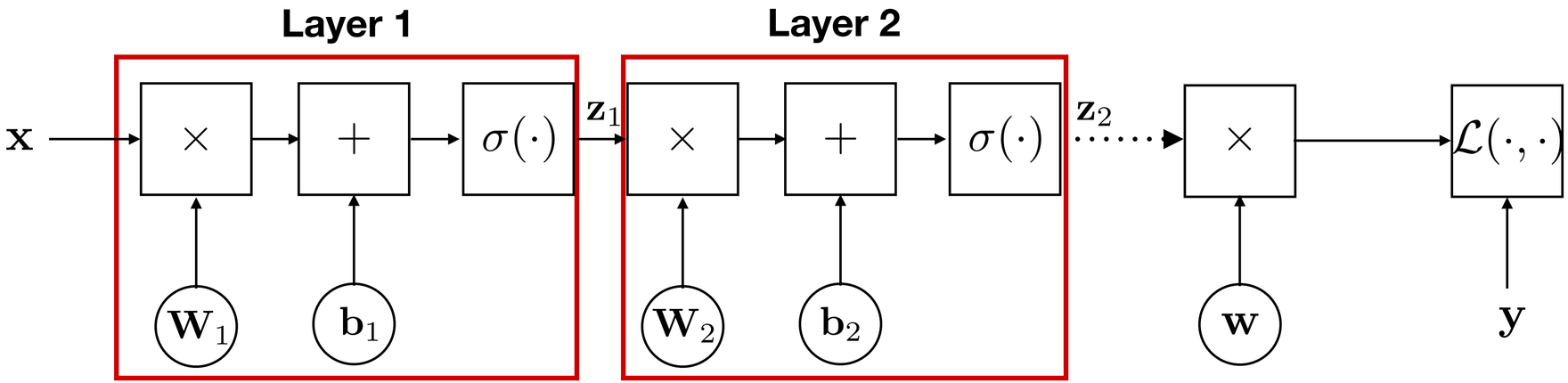
$$\mathbf{w}_t = \mathbf{w}_{t-1} - \frac{\lambda_t \boldsymbol{\mu}_t}{\sqrt{\boldsymbol{\nu}_t + \epsilon}}$$

Note that ADAM also uses a running estimate debiasing step, not shown here.

Large noise, small learning rate

$$\boldsymbol{\nu}_t \simeq \boldsymbol{\mu}_t^2 + \boldsymbol{\sigma}_t^2$$
$$\mathbb{E}[\tilde{\mathbf{g}}_t] \simeq \boldsymbol{\mu}_t$$
$$\Rightarrow \frac{\tilde{\mathbf{g}}_t}{\sqrt{\nu}_t} \simeq \frac{\boldsymbol{\mu}_t}{\sqrt{\boldsymbol{\mu}_t^2 + \boldsymbol{\sigma}_t^2}} \leq 1$$

The vanishing—exploding gradient problem



Assume

$$w_{ij} \sim \mathcal{N}(w_{ij}|0, \tau^2)$$
$$x_i \sim \mathcal{N}(x_i|0, \sigma^2)$$
$$\mathbb{E}[x_i x_j] = 0$$

Let's consider the variance of activations $\mathbf{x}$ in the forward pass. For small activations, we operate in a near-linear regime $\mathbf{z} \simeq \mathbf{W}\mathbf{x}$

$$\mathbb{V}_{\mathbf{z}}[\mathbf{z}] \simeq \mathbb{V}_{\mathbf{W},\mathbf{x}}[\mathbf{W}\mathbf{x}] = \mathbb{E}_{\mathbf{W},\mathbf{x}}\left[\mathbf{W}\mathbf{x}\mathbf{x}^\top\mathbf{W}^\top\right] = \mathbb{E}_{\mathbf{W}}\left[\mathbf{W}\underbrace{\mathbb{E}_{\mathbf{x}}\left[\mathbf{x}\mathbf{x}^\top\right]}_{\sigma^2\mathbf{I}}\mathbf{W}^\top\right] = \sigma^2\underbrace{\mathbb{E}_{\mathbf{W}}\left[\mathbf{W}\mathbf{W}^\top\right]}_{n\tau^2\mathbf{I}} = n\sigma^2\tau^2\mathbf{I}$$

dimensions

Can you prove this?

Variance grows/decays exponentially in depth

$$\mathbb{V}[\mathbf{z}_\ell] \simeq (n\tau^2)^\ell \mathbb{V}[\mathbf{x}]$$

$n\tau^2 > 1$ Activation explosion

$n\tau^2 < 1$ Activation vanishing

Initialization

Activations

Fan in

$$\mathbb{V}[\mathbf{z}_{\ell+1}] \simeq n_\ell \tau_\ell^2 \mathbb{V}[\mathbf{z}_\ell]$$

Make equal $n_\ell \tau_\ell^2 = 1$

Gradients

Fan out

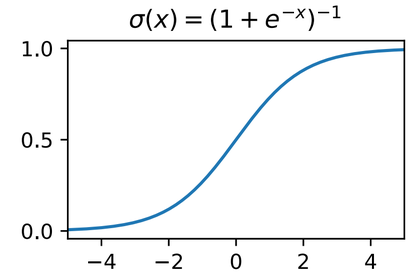
$$\mathbb{V}\left[\frac{\partial \mathcal{L}}{\partial \mathbf{z}_{\ell+1}}\right] \simeq n_{\ell+1} \tau^2 \mathbb{V}\left[\frac{\partial \mathcal{L}}{\partial \mathbf{z}_\ell}\right]$$

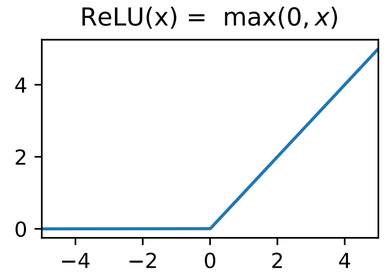
$n_{\ell+1} \tau_\ell^2 = 1$ **Make equal**

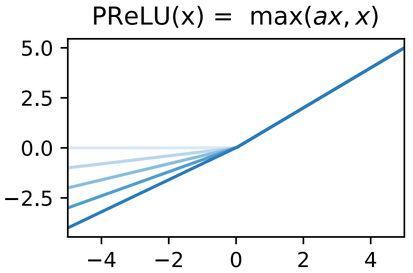
Xavier/Glorot initialization $\tau_\ell^2 = \dfrac{2}{n_\ell + n_{\ell+1}}$

Kaiming/He initialization $\tau_\ell^2 = \dfrac{2}{n_\ell}$

$$\tau_\ell^2 = \frac{2}{(1+a^2) n_\ell}$$







Activation Normalization

Initialization helps at the start of training, but how do we maintain stability **during** training?

Data whitening

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$$

$$\mathbf{\Sigma} = \mathbb{E}\left[(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^\top\right]$$

Find $\mathbf{z} = \mathbf{W}(\mathbf{x} - \bar{\mathbf{x}})$ such that

$$\mathbb{V}[(\mathbf{z} - \bar{\mathbf{z}})(\mathbf{z} - \bar{\mathbf{z}})^\top] = \mathbf{I}$$

$$\mathbf{W}\mathbf{\Sigma}\mathbf{W}^\top = \mathbf{I}$$

Eigendecomposition $\mathbf{\Sigma} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$

Solutions

$\mathbf{W} = \mathbf{\Lambda}^{-1/2}\mathbf{U}^\top$ PCA whitening

$\mathbf{W} = \mathbf{U}\mathbf{\Lambda}^{-1/2}\mathbf{U}^\top$ ZCA whitening

$\mathbf{W} = \mathbf{\Sigma}^{1/2}$ Cholesky whitening

Cholesky decomposition

Batch normalization

$$z_i = \frac{x_i - \bar{x}_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

$\epsilon \ll 1$

Equivalent to data whitening with diagonal $\mathbf{W}$

Learnable affine transformation included

$$y_i = \gamma_i z_i + \beta_i$$

The weird bit

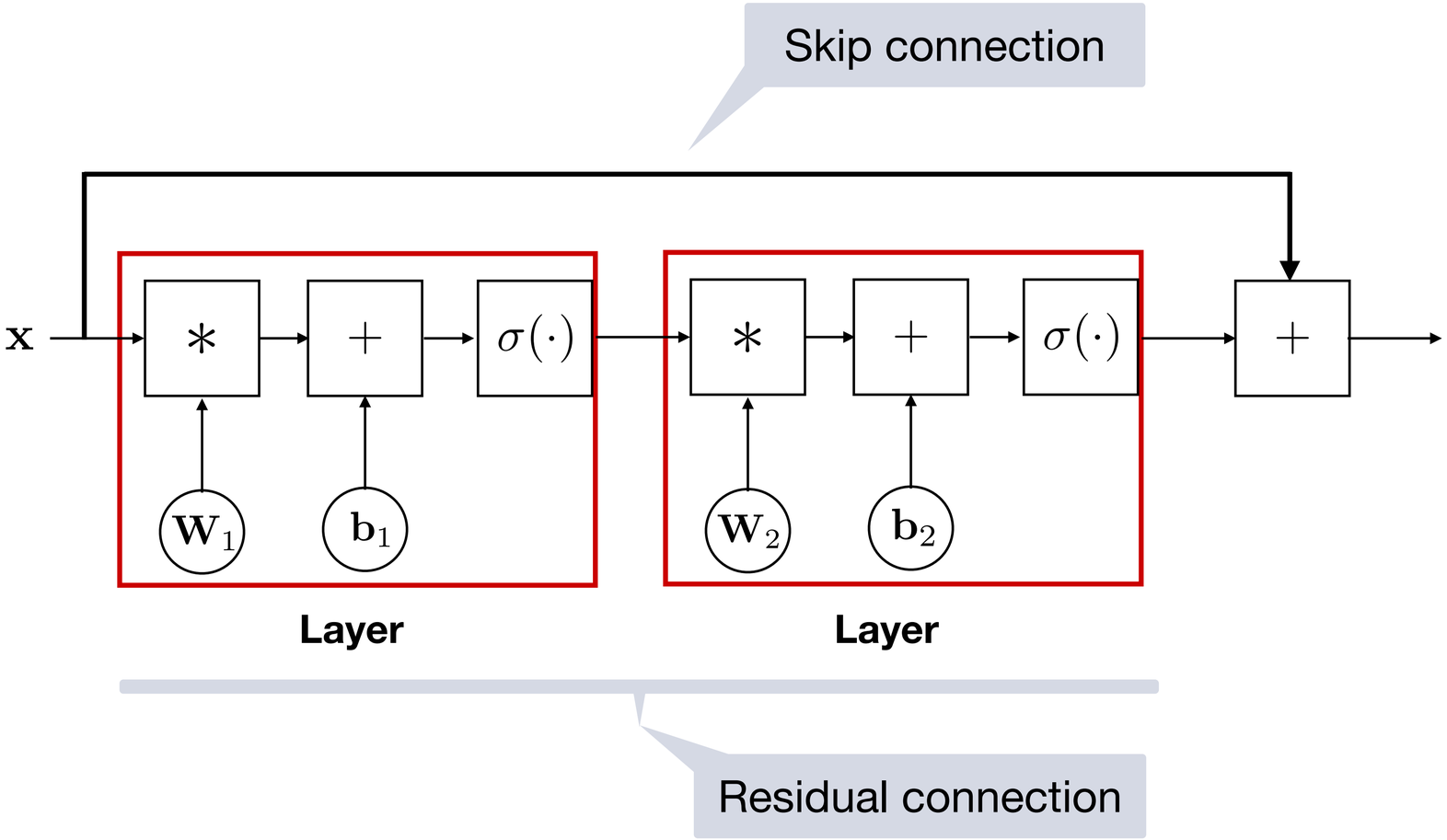
Training: $\bar{x}_i$ and σ_i^2 computed per minibatch

Test: $\bar{x}_i$ and σ_i^2 based on exponential moving averages collected during training

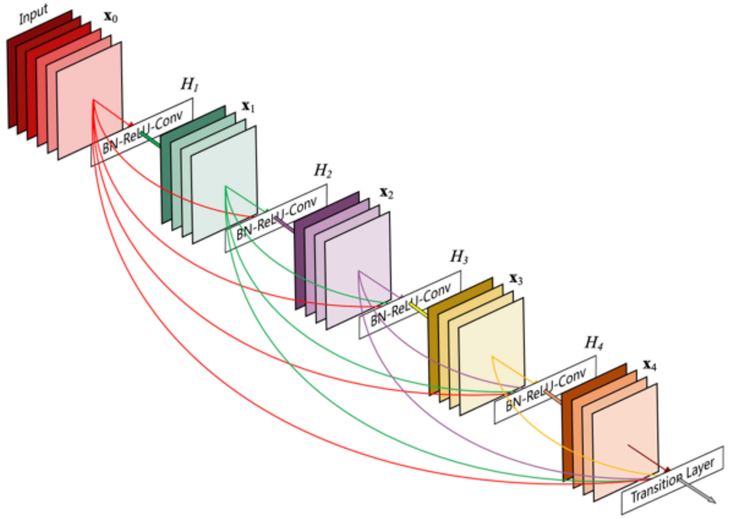
Why does it work? Some say, gradient normalization, others loss landscape

Skip connections

A popular technique to combat vanishing gradients is *skip connections*



Famously found in ResNets and DenseNets



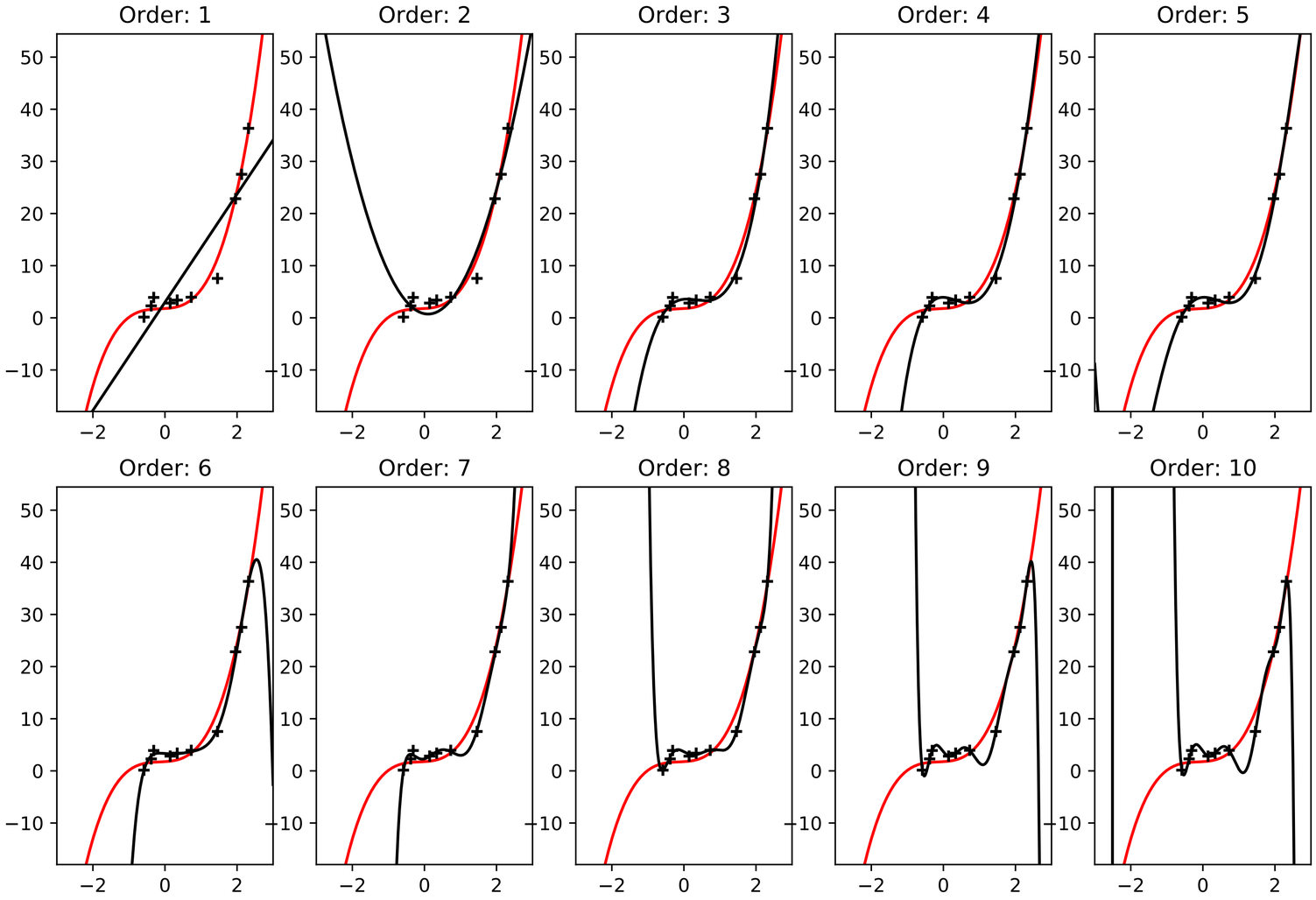
Can build networks 100s of layers deep!

Learning Theory

Pantai Ora, Maluku

Overfitting

How do we decide the order of polynomial features?



Higher orders can fit the data perfectly, but fail dramatically on new *test data*

Generalization Error

The notion we are describing is called *generalization error*

$$\mathcal{R}(f) = \int L(f(\mathbf{x}), \mathbf{y}) \mathrm{d}p_*(\mathbf{x}, \mathbf{y})$$

Supervised learning then boils down to minimizing generalization error

$$f^* = \arg\min_{f \in \mathcal{F}} \mathcal{R}(f)$$

Due to lack of access, we use the *empirical risk*

$$\hat{\mathcal{R}}(f) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} L(f(\mathbf{x}), \mathbf{y})$$

Hopefully this looks familiar, the negative log-likelihood is an example!

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^{N} \log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$$

$$\int f(\mathbf{x}) \mathrm{d}p(\mathbf{x}) \overset{\text{either}}{=} \int f(\mathbf{x}) p(\mathbf{x}) \, \mathrm{d}\mathbf{x} \overset{\text{or}}{=} \sum_{\mathbf{x}} f(\mathbf{x}) p(\mathbf{x})$$

Overfitting: pathological behavior fitting statistical aberrations in $\mathcal{D}$, not in p_*

Regularized risk minimization

$$f_\lambda = \arg\min_f [\hat{\mathcal{R}}(f) + \lambda C(f)]$$

Regularizer: if -log(prior), this is MAP!

Structural risk minimization

$$\lambda^* = \arg\min_\lambda \hat{\mathcal{R}}(f_\lambda)$$

Risk estimate, typically cross-validation (coming later)

Bias—variance decomposition

If L is MSE, we can understand the expected error

$$\mathbb{E}_{\mathcal{D}}[(y - \underbrace{f_{\mathcal{D}}(\mathbf{x})}_{\text{pred.}})^2] = \mathbb{E}_{\mathcal{D}}\left[(y - \bar{f}(\mathbf{x}) + \bar{f}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x}))^2\right]$$

$$= \underbrace{(y - \bar{f}(\mathbf{x}))^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[(f_{\mathcal{D}}(\mathbf{x}) - \bar{f}(\mathbf{x}))^2\right]}_{\text{variance}}$$

$$+ \underbrace{2(y - \bar{f}(\mathbf{x}))\mathbb{E}_{\mathcal{D}}\left[\bar{f}(\mathbf{x}) - f_{\mathcal{D}}(\mathbf{x})\right]}_{=0}$$

$$\bar{f}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}[f_{\mathcal{D}}(\mathbf{x})]$$

Optimal model minimizes both variance and bias!

When you include the expectation over $\mathbf{x}$ we get

$$\mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_{\mathbf{x},y|\mathcal{D}}[(y - f_{\mathcal{D}}(\mathbf{x}))^2]\right] = \text{bias}^2 + \text{variance} + \text{intrinsic noise}$$

Aleatoric noise

- Cause: data generation noise
- Independent of model

Underfit term

- Cause: model inflexibility
- Found in rigid models

Overfit term

- Cause: spurious sampling
- Found in flexible models

Bias—variance decomposition

Groundtruth: cubic

Model: quartic with Gaussian likelihood and prior (MAP parameters)

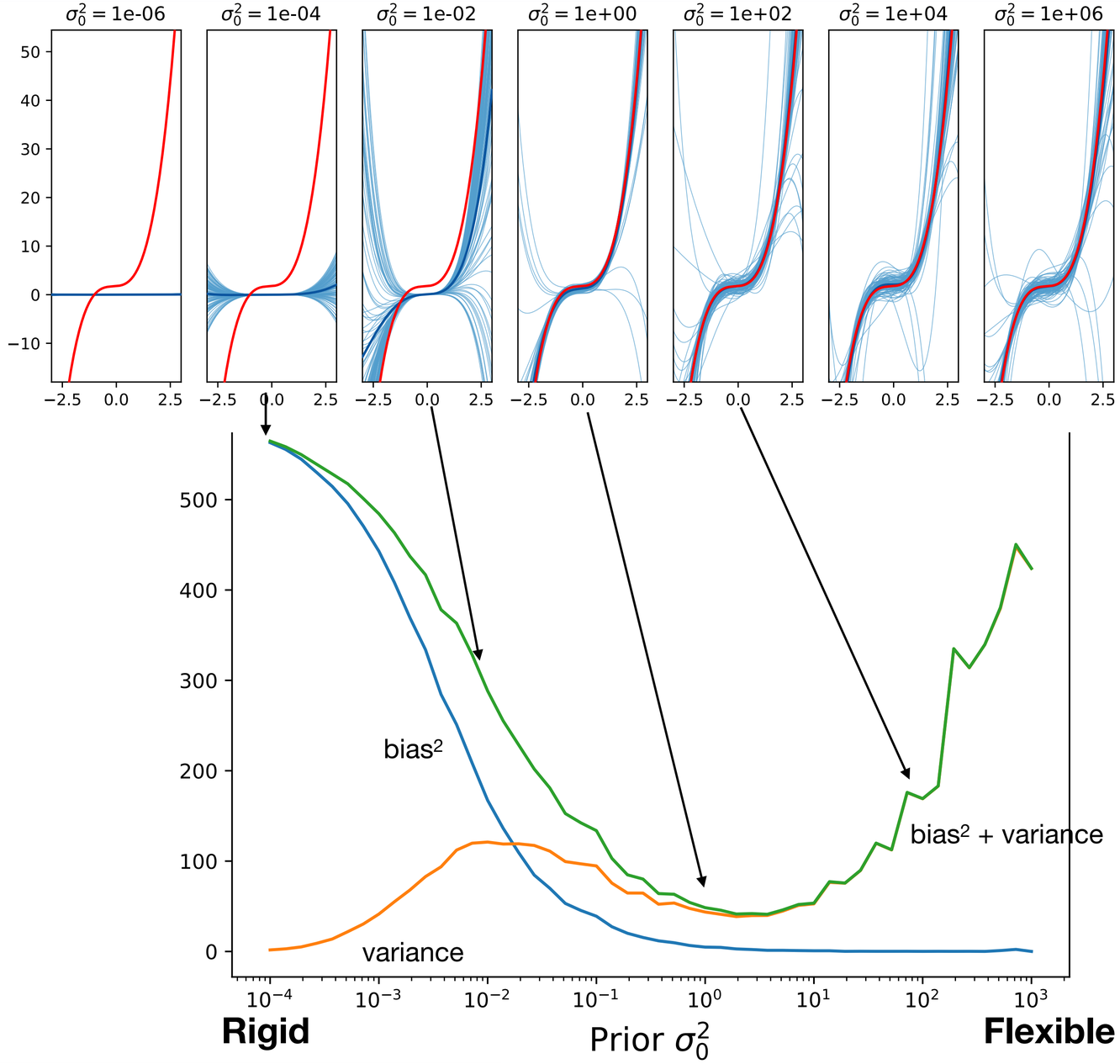
$$\arg\max_{\mathbf{w}} p(\mathcal{D}|\mathbf{w})p(\mathbf{w}|\sigma_0^2)$$

Num runs: 1000

Bias² decreases with looser prior

Variance increases with looser prior

Sum of bias² and variance has minimum at minimum test error



Cross-validation

How do we design networks in practice?

✗ Generalization error: cannot measure

✗ Choose a prior: how?

✗ Bias—variance tradeoff: assumes we have an infinite number of datasets

The most common technique is *cross-validation*

1. Train on $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$

2. Test on *held-out validation set*

$$\mathcal{V} = \{(\mathbf{x}_{N+1}, y_{N+1}), ..., (\mathbf{x}_{N+M}, y_{N+M})\}$$

3. Repeat until you run out of AWS credits

Pick best architecture on validation set

The notion we are describing is called *generalization error*

$$\mathcal{R}(f) = \int L(f(\mathbf{x}), \mathbf{y}) \mathrm{d}p_*(\mathbf{x}, \mathbf{y})$$

Supervised learning then boils down to minimizing generalization error

$$f^* = \arg\min_{f \in \mathcal{F}} \mathcal{R}(f)$$

Due to lack of access, we use the *empirical risk*

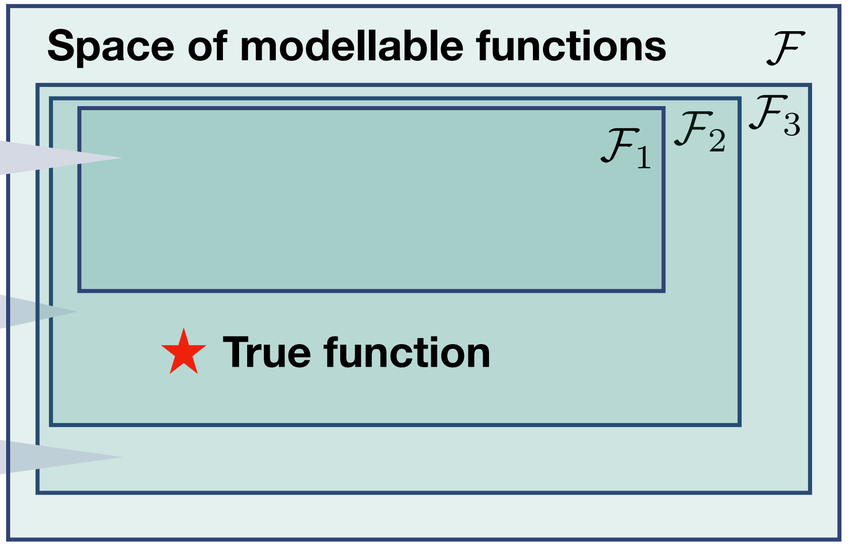
$$\mathcal{R}(f) \simeq \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} L(f(\mathbf{x}), \mathbf{y})$$

Overfitting: pathological behavior fitting statistical aberrations in $\mathcal{D}$, not in p_*

Inductive bias

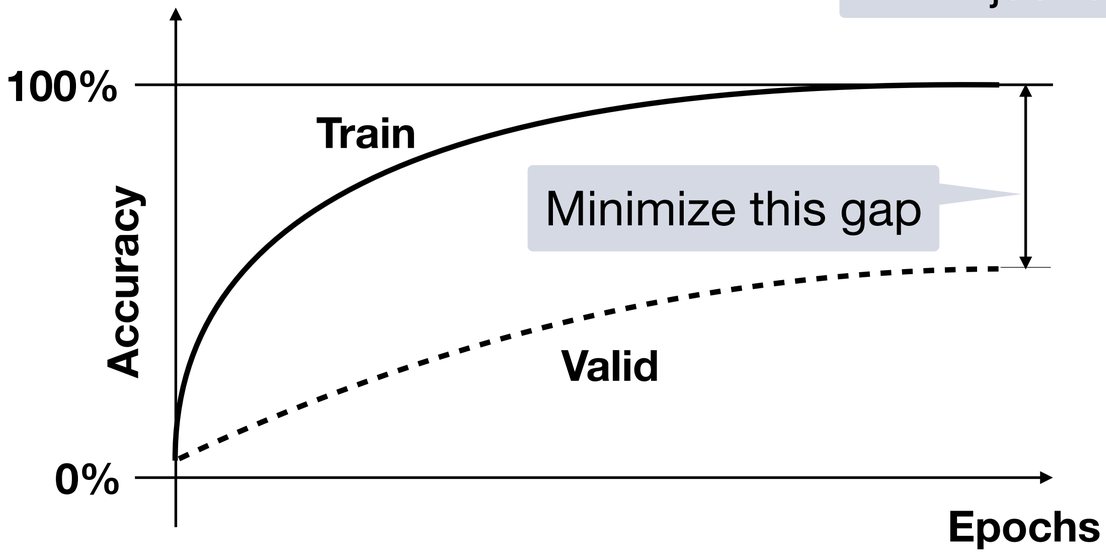
One cross-validation procedure

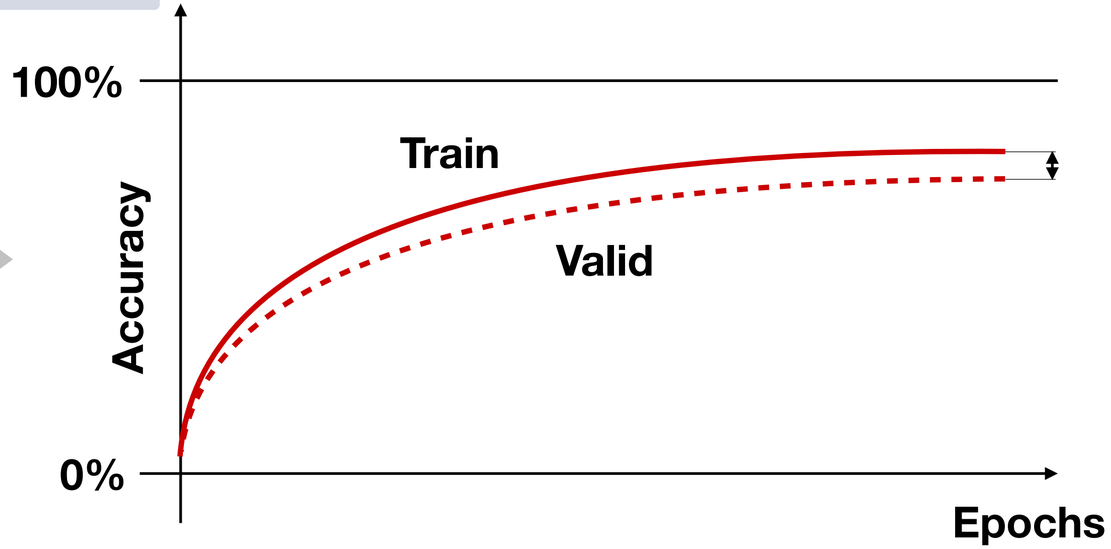
1. Overfit on the training set
2. Increase model bias to balance bias^2 and variance
3. Stop at smallest generalization gap



$$f^* = \underset{f \in \mathcal{F}_i}{\arg\min}\, \mathcal{R}(f)$$

Choose function class with just enough bias





Neural Architecture

Istana Pagaruyung, West Sumatera

Convolutions

For images we use *convolutions*.

Convolutions use:
1. Translational weight-tying
2. Small reception fields

Drop-in replacement for the linear layer.

Indeed, it is just a linear layer!

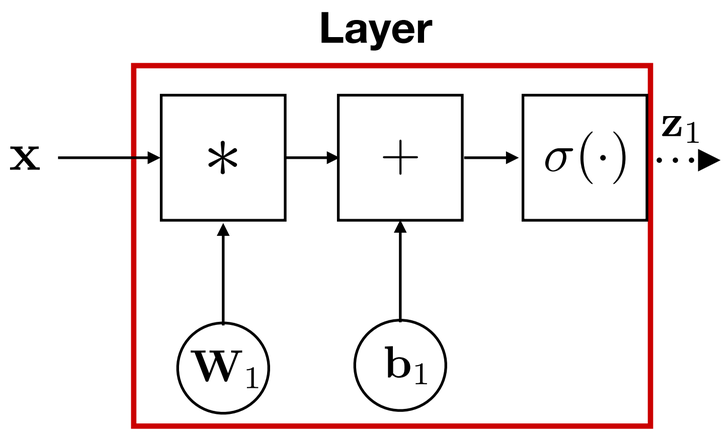
Standard convolution

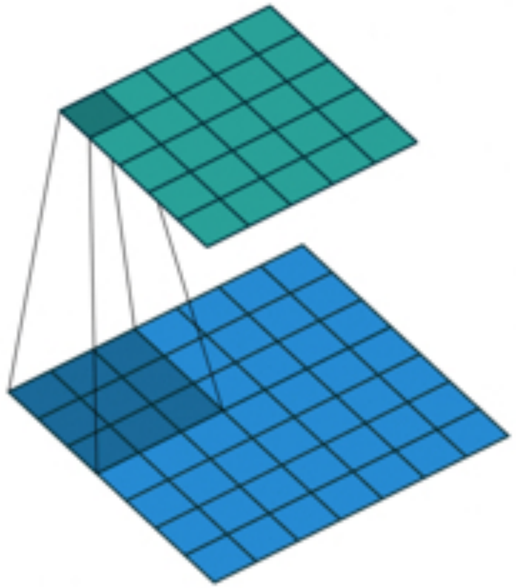
$$[\mathbf{F} * \mathbf{W}](\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \mathbf{F}(\mathbf{y}) \mathbf{W}(\mathbf{y} - \mathbf{x})$$

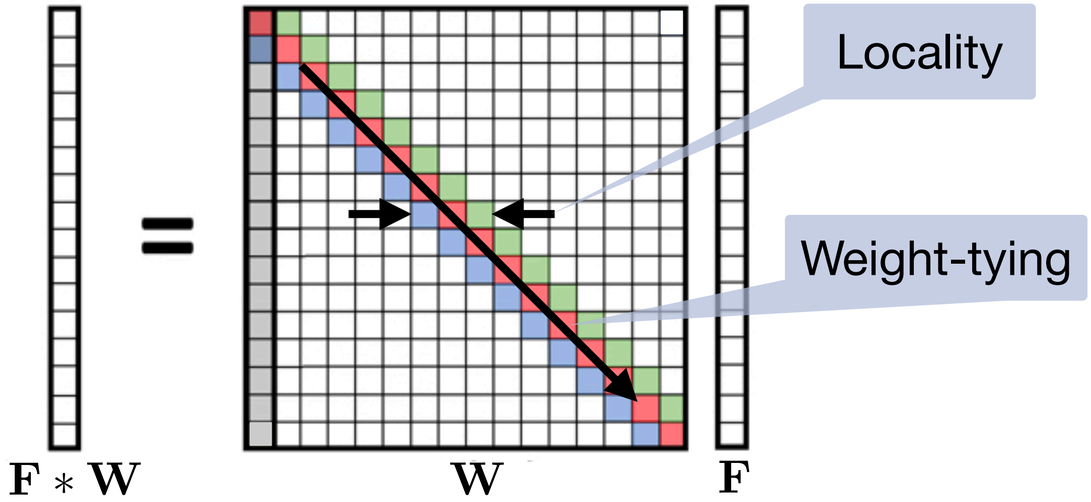
Filter

Feature map

e.g. LeCun et al. (1991)







Convolutions can be “reshaped” into matrix-vector products, revealing the weight-tying and locality

How to derive convolution from first principles, Michael Bronstein

Convolutions

Each layer in a *convolutional neural network* (CNN) consists of multiple channels

$$[\mathbf{F} * \mathbf{W} + \mathbf{b}]_k(\mathbf{x}) = b_k + \sum_{c=1}^{C} \sum_{\mathbf{y} \in \mathbb{Z}^2} \mathbf{F}(\mathbf{y}) \mathbf{W}_{c,k}(\mathbf{y} - \mathbf{x})$$

Output channel

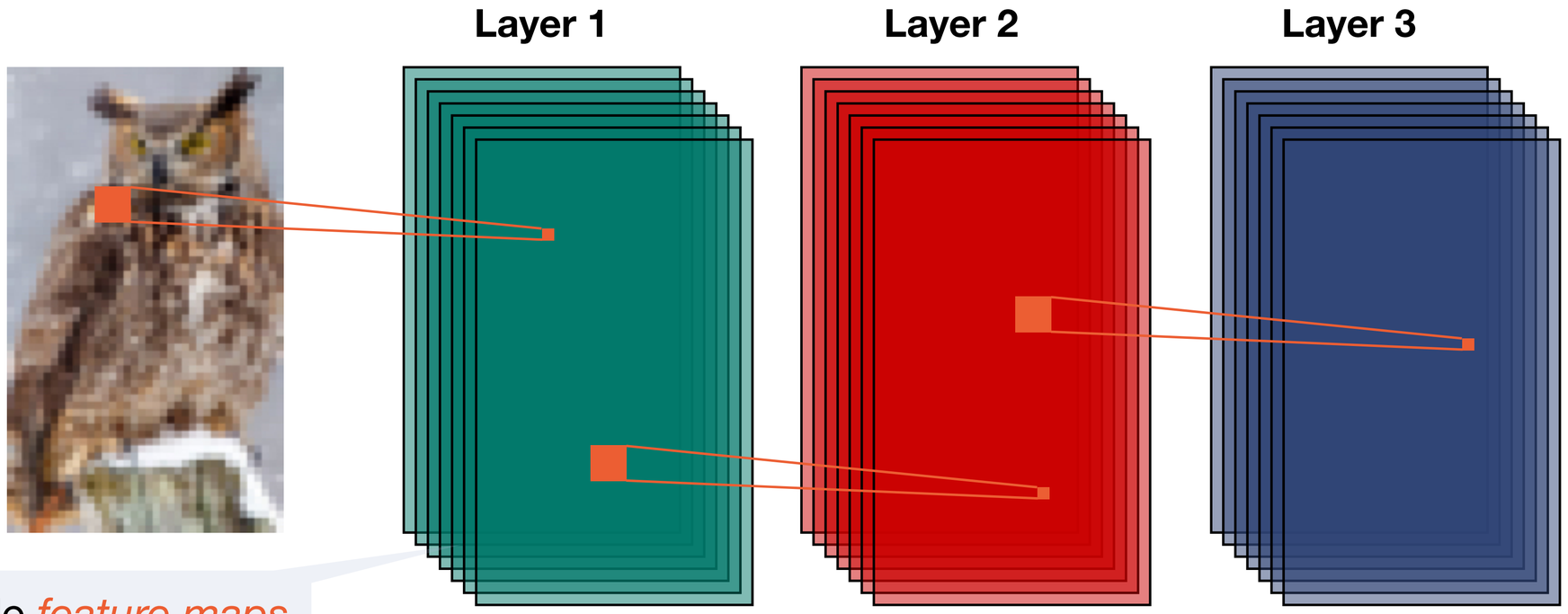
Bias

Input channel

Number of learnable parameters $\mathcal{O}(CKd^2)$

down from $\mathcal{O}(CKD^2)$

where $d \ll D$



Multiple *feature maps*

How to derive convolution from first principles, Michael Bronstein

Pooling

Pooling combined with *striding* reduces the size of feature maps.

This is desirable for:

- Computational constraints
- Empirical performance increase
- Increased small deformation invariance

Max pooling

$$\text{max-pool}[\mathbf{F}](\mathbf{x}) = \max_{\mathbf{x}' \in \mathcal{N}_{\mathbf{x}}} \mathbf{F}(\mathbf{x}')$$

Average pooling

$$\text{average-pool}[\mathbf{F}](\mathbf{x}) = \frac{1}{|\mathcal{N}_{\mathbf{x}}|} \sum_{\mathbf{x}' \in \mathcal{N}_{\mathbf{x}}} \mathbf{F}(\mathbf{x}')$$

Neighborhood

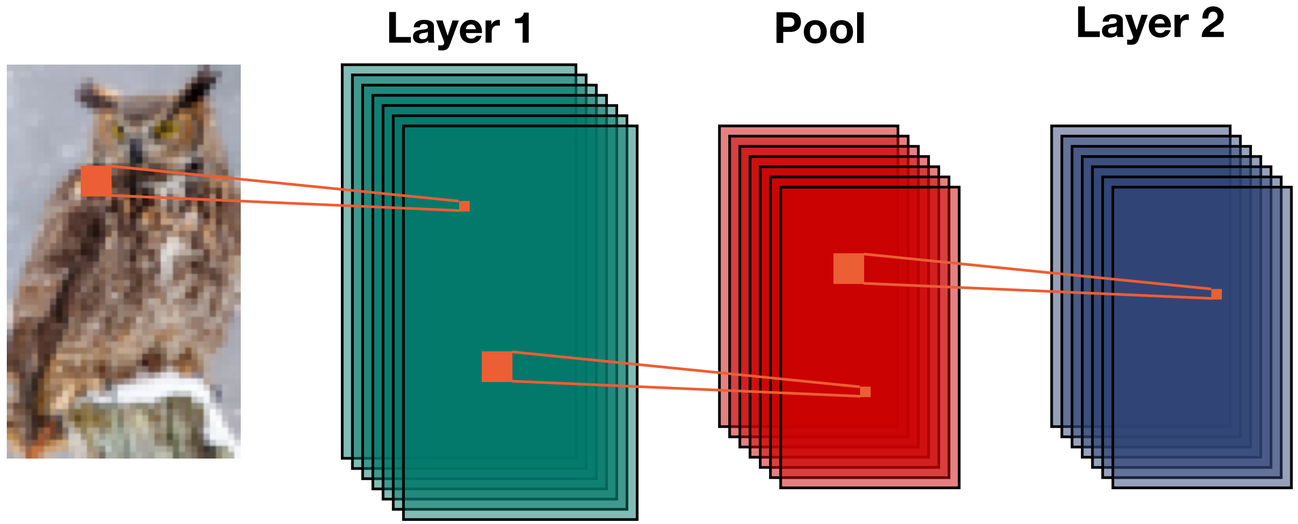
Standard convolution

$$[\mathbf{F} * \mathbf{W}](\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \mathbf{F}(\mathbf{y}) \mathbf{W}(\mathbf{y} - \mathbf{x})$$

Stride-n convolution

$$[\mathbf{F} *_n \mathbf{W}](\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \mathbf{F}(\mathbf{y}) \mathbf{W}(\mathbf{y} - n\mathbf{x})$$

Integer



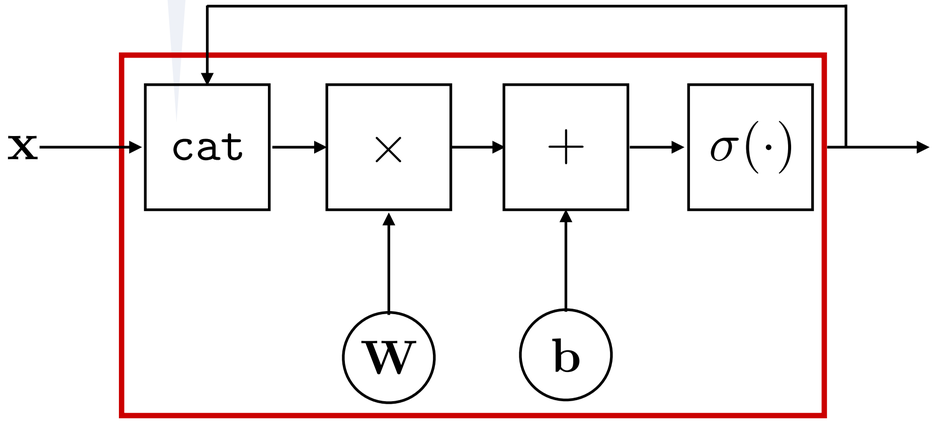
Recurrent Neural Networks

For *time-series data* use a *recurrent neural network* (RNN)

Example loss function

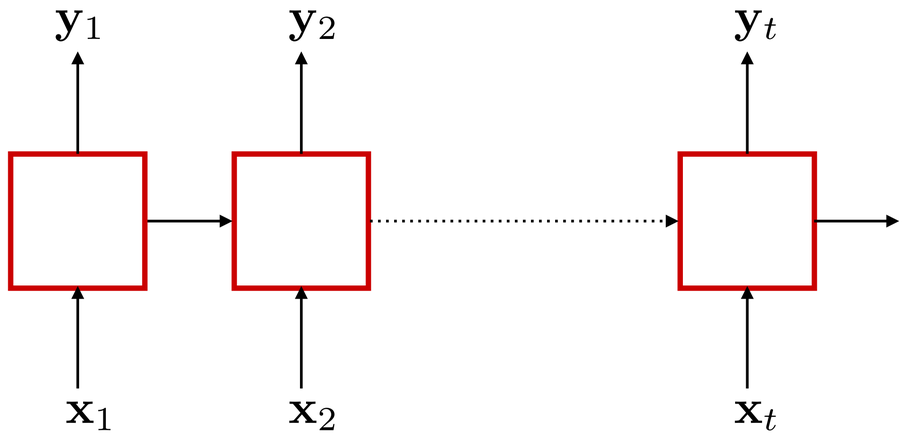
$$\mathcal{L}(\mathbf{W}; \mathbf{x}_{1:t}) = \sum_{i=1}^{t} \mathcal{L}_i(\mathbf{y}_i, f(\mathbf{x}_{1:i}))$$

Concatenation: $\texttt{cat}(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$



Recurrent Layer

Unrolling in time



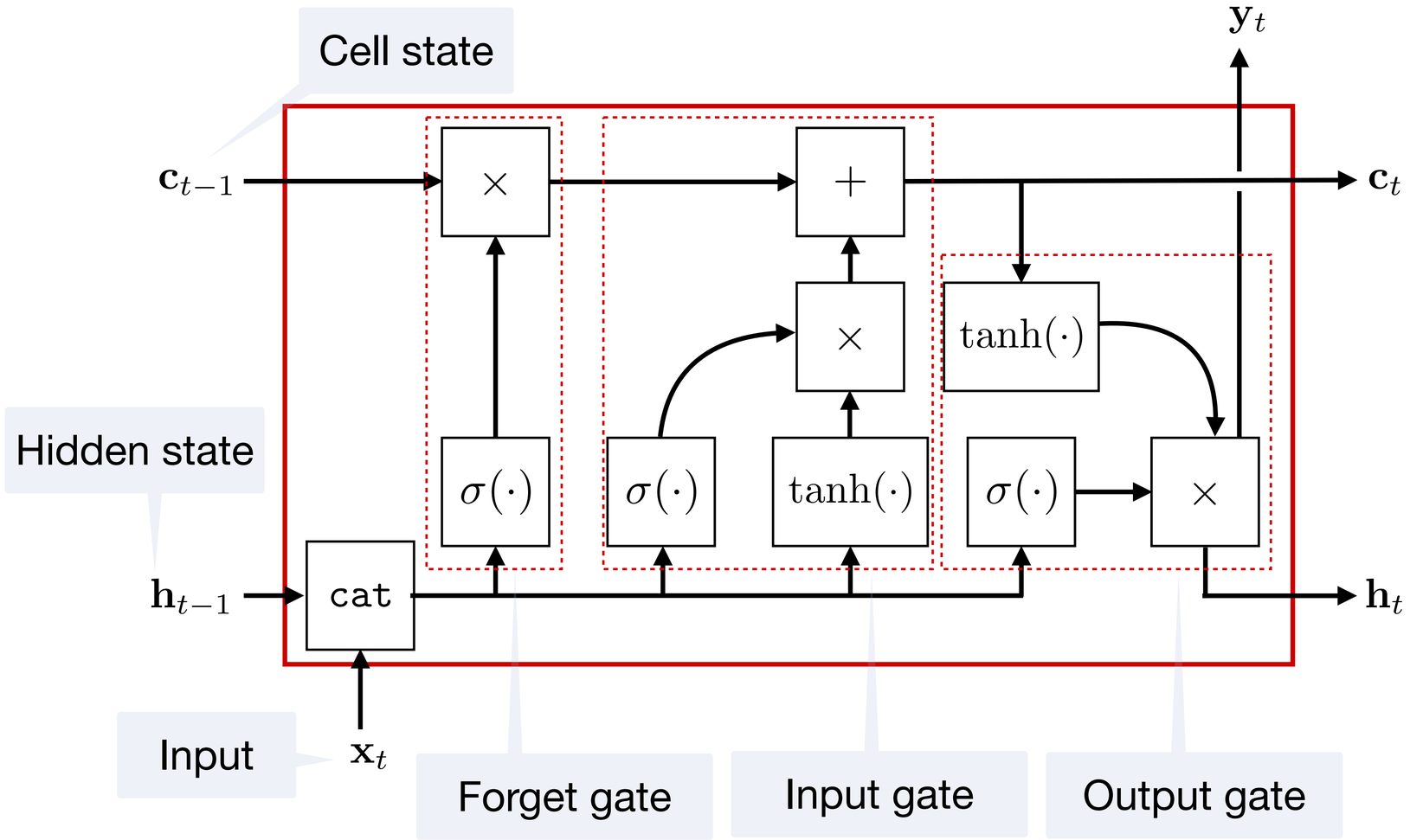
We can see that for long sequences, the activations/gradients will suffer from the exploding/vanishing gradient problem

Long Short—Term Memory Networks

A stabler recurrent layer is the *long short—term memory layer* (LSTM).

A vast array of variants exist:

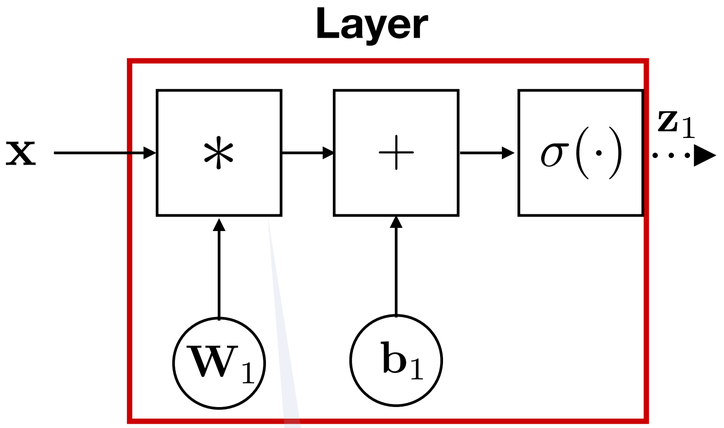
- Gated recurrent unit (GRU)
- Bidirectional LSTM
- Neural ODE

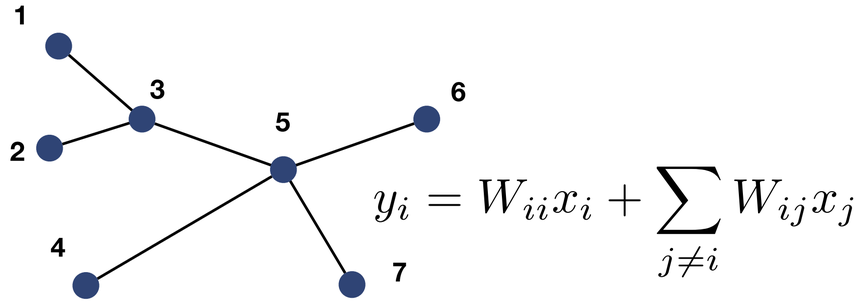


Graph Neural Networks

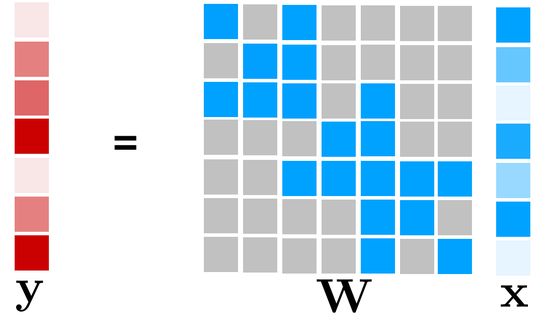
A common domain is graphs.
Examples are:

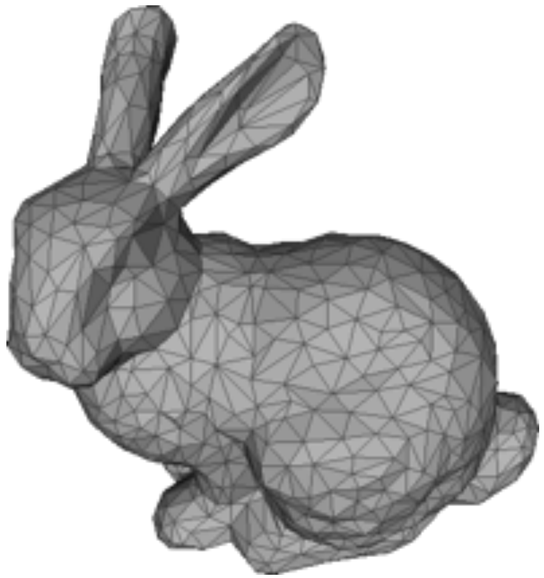
- Social networks
- Molecular structures
- Meshes

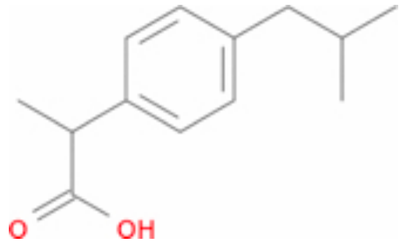


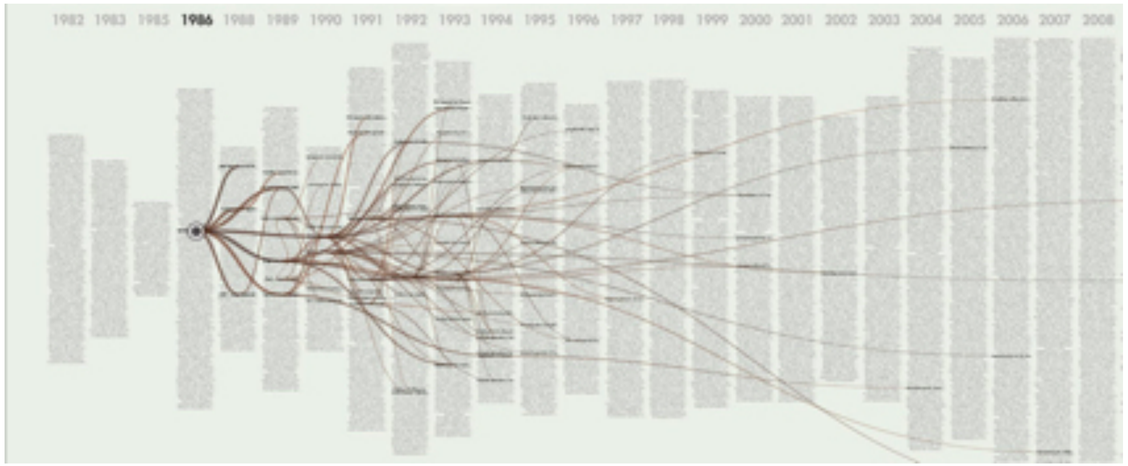


Graph convolution

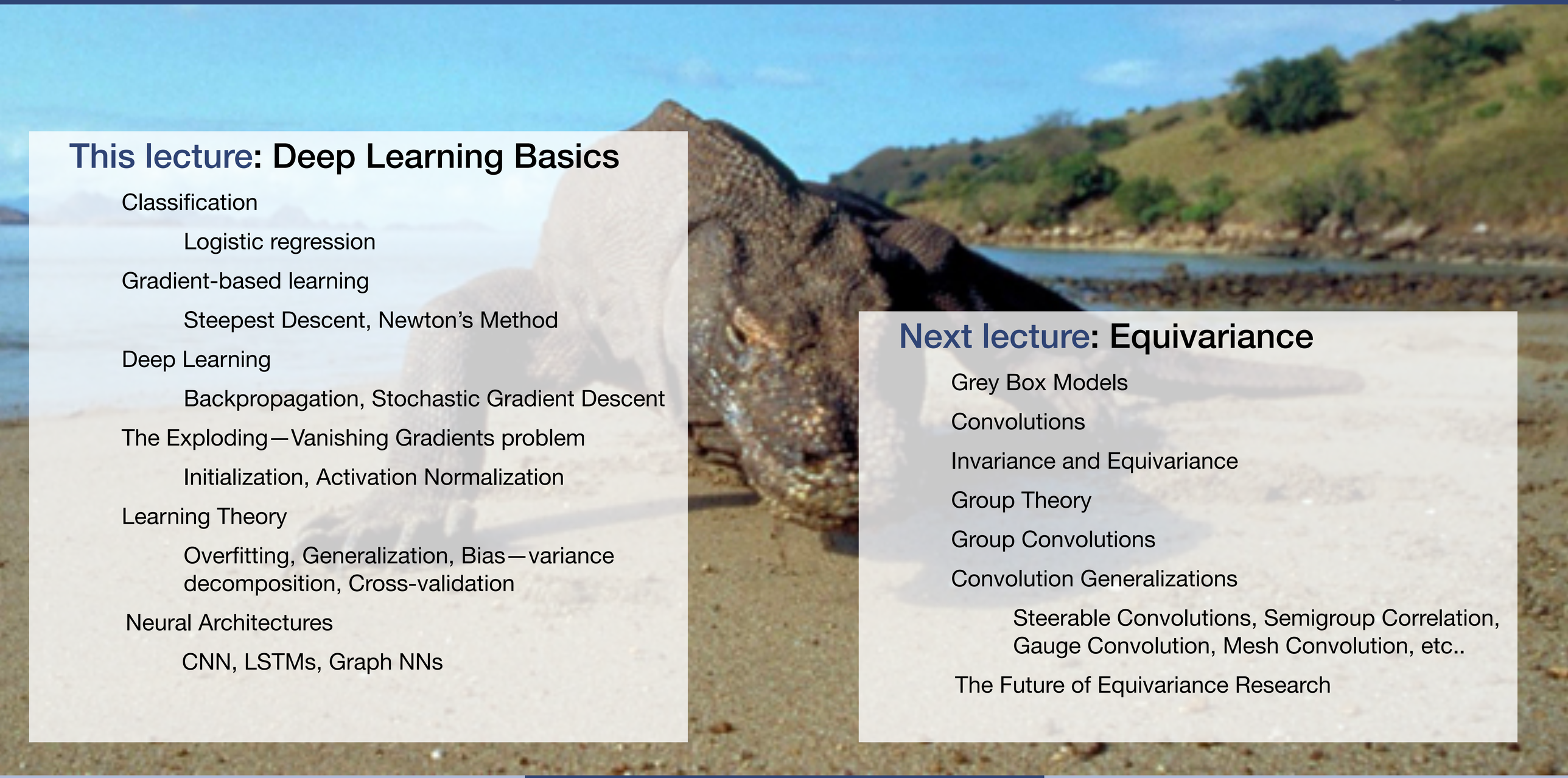








Summary



This lecture: Deep Learning Basics

- Classification
  - Logistic regression
- Gradient-based learning
  - Steepest Descent, Newton's Method
- Deep Learning
  - Backpropagation, Stochastic Gradient Descent
- The Exploding—Vanishing Gradients problem
  - Initialization, Activation Normalization
- Learning Theory
  - Overfitting, Generalization, Bias—variance decomposition, Cross-validation
- Neural Architectures
  - CNN, LSTMs, Graph NNs

Next lecture: Equivariance

- Grey Box Models
- Convolutions
- Invariance and Equivariance
- Group Theory
- Group Convolutions
- Convolution Generalizations
  - Steerable Convolutions, Semigroup Correlation, Gauge Convolution, Mesh Convolution, etc..
- The Future of Equivariance Research