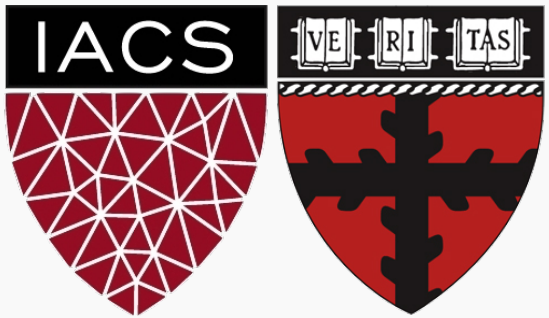
Advanced Section #2: Understanding dropout, optimization algorithms, batch normalization

CS 209B: Data Science

Javier Zazo Pavlos Protopapas



Lecture Outline

Bias and variance

Dropout

Batch normalization

Optimization algorithms

Gradient checking

Bias and variance

Estimators

- Point estimation is the attempt to provide the single "best" prediction of some quantity of interest:

$$\hat{\boldsymbol{\theta}}_m = g(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}).$$

  - $\boldsymbol{\theta}$: true value.
  - $\hat{\boldsymbol{\theta}}_m$: estimator for m samples.
- Frequentist perspective: we assume the true parameter $\boldsymbol{\theta}$ is fixed but unkwown.
- Since the data is drawn from a random process, any function of the data is random $\implies$ $\hat{\boldsymbol{\theta}}_m$ is a r.v.

Bias and Variance

- Bias of an estimator:

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbb{E}(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta}$$

- Variance of an extimator: $\text{Var}(\hat{\boldsymbol{\theta}}_m)$.

Examples:

  - Sample mean: $\hat{\mu}_m = \frac{1}{m}\sum_i \mathbf{x}^{(i)}$
  - Sample variance $\hat{\sigma}_m^2 = \frac{1}{m}\sum_i (x^{(i)} - \hat{\mu}_m)^2$:

$$\mathbb{E}[\hat{\sigma}_m^2] = \frac{m-1}{m}\sigma^2$$

  - Unbiased sample variance: $\tilde{\sigma}_m^2 = \frac{1}{m-1}\sum_i (x^{(i)} - \hat{\mu}_m)^2$

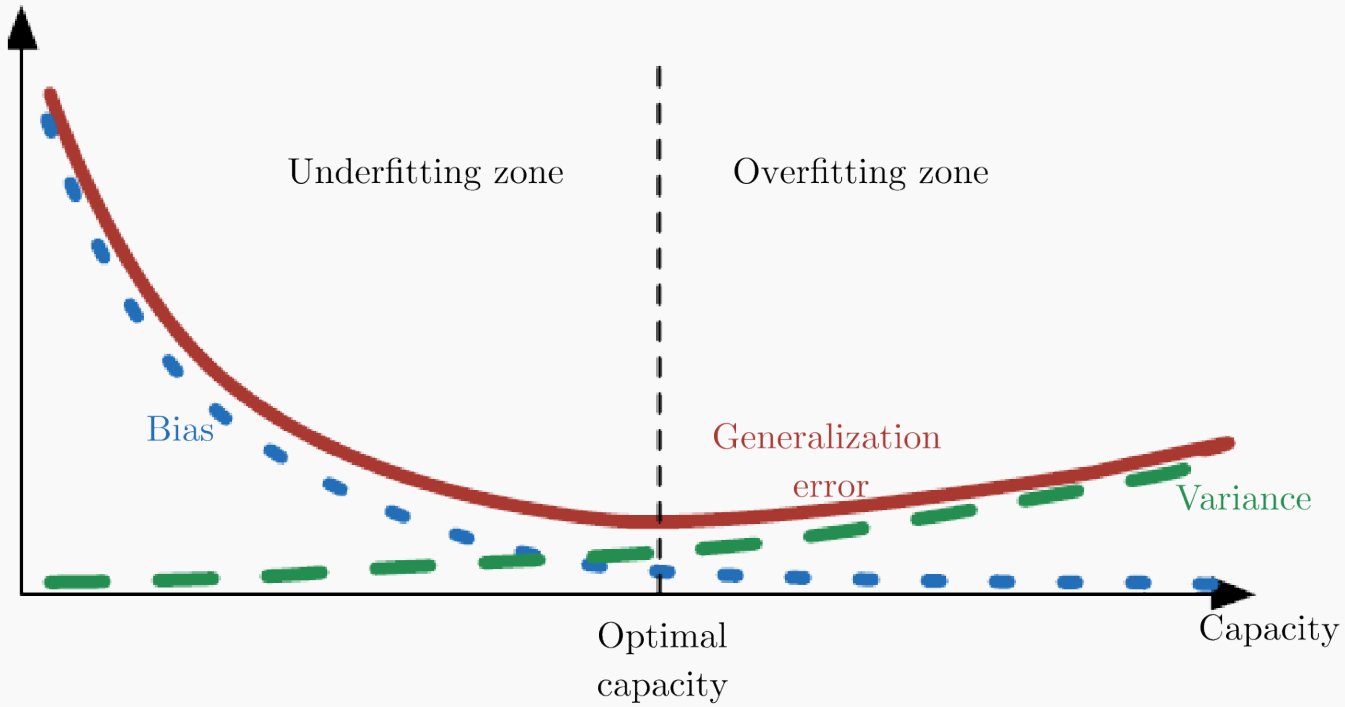
Bias-Variance trade-off (I)

- Bias and variance measure two different sources of error in an estimator.
  - Bias: expected deviation from the true value.
  - Variance: deviation from the expected estimator that any particular sampling of the data is likely to cause.
- How to choose estimators with different statistics?
  - Use cross-validation: highly successful (empirical).
  - Use mean square error (MSE):

$$\begin{aligned} \text{MSE} &= \mathbb{E}[(\hat{\theta}_m - \theta)^2] \\ &= \text{bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m) \end{aligned}$$

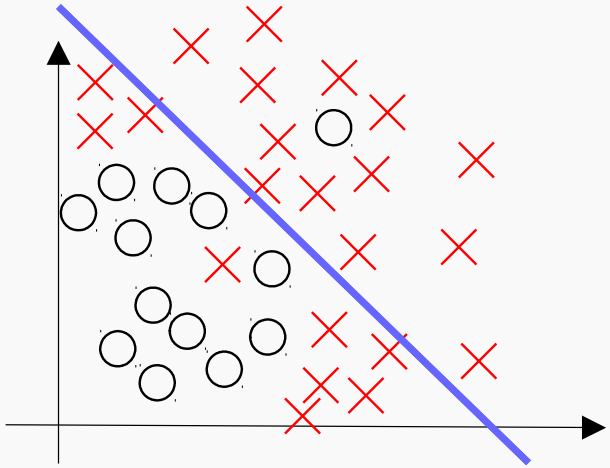
Bias-Variance trade-off (II)

- There is a trade-off between bias and variance.
- It is directly related to the two central challenges of ML: overfitting and underfitting.

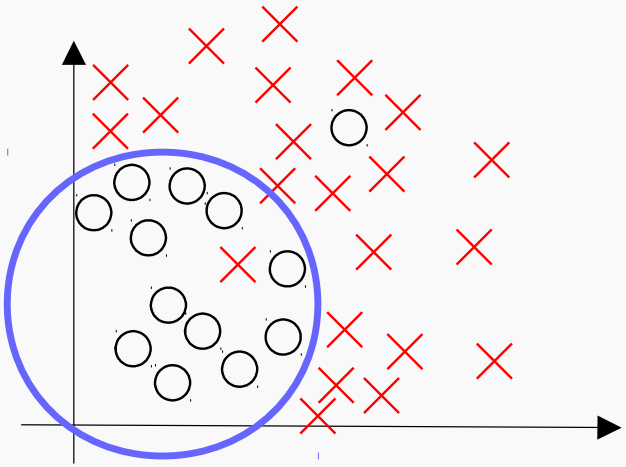


Source: Goodfellow et. al

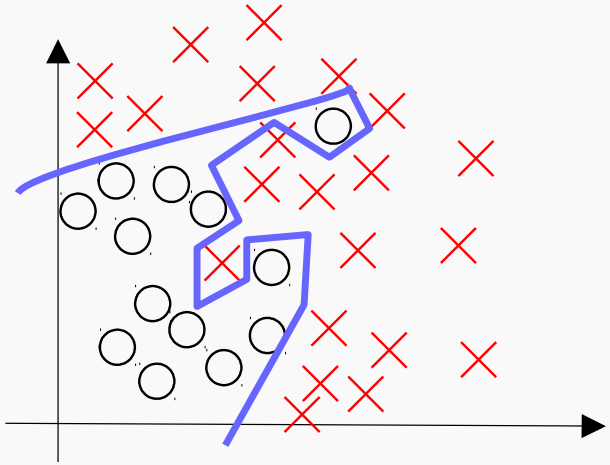
Bias-Variance Example



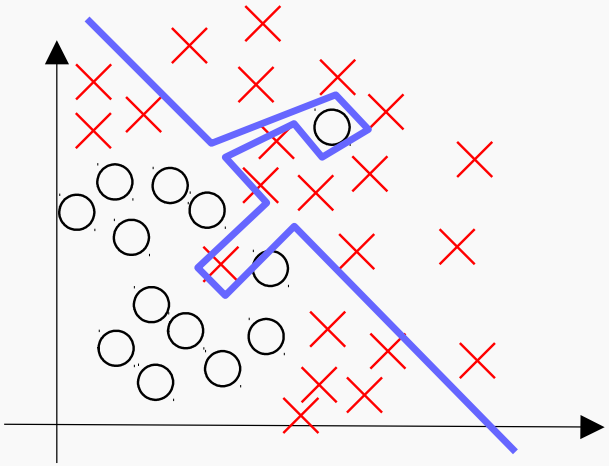
high bias &
underfitting



appropriate



high variance &
overfitting



high bias & variance

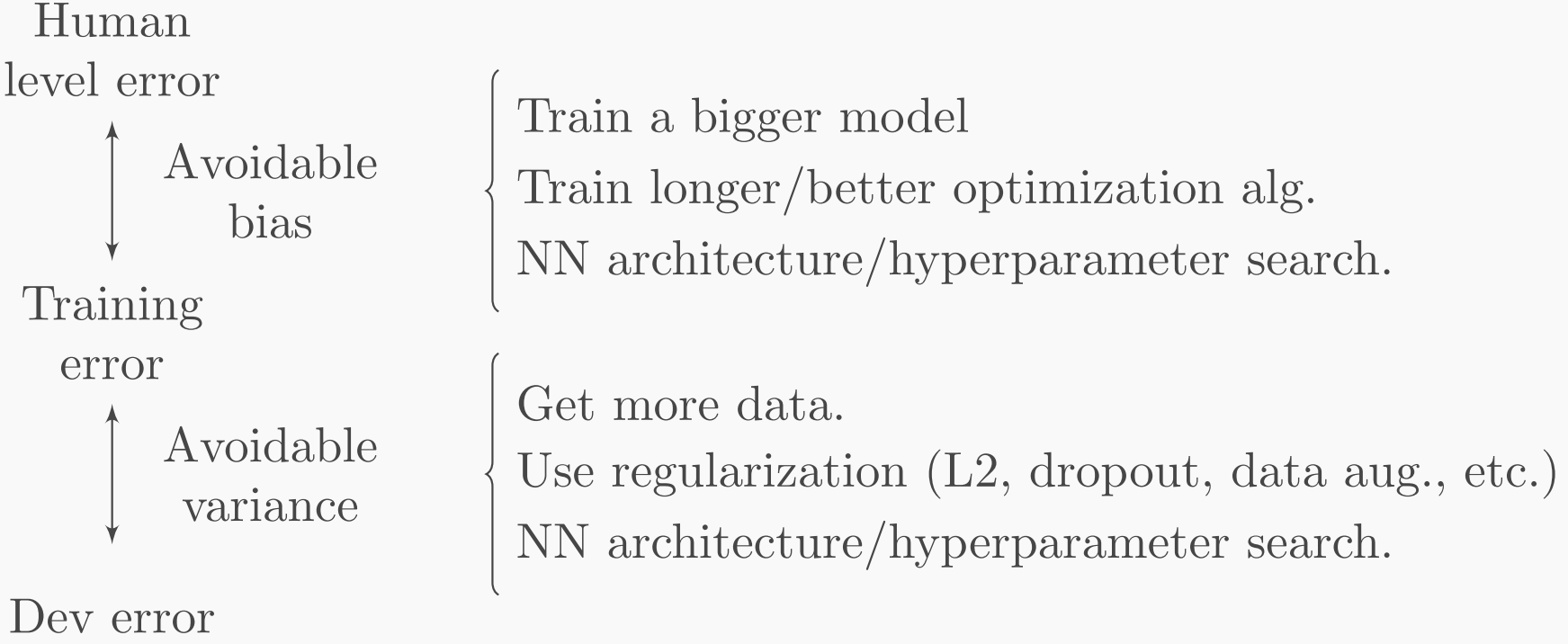
Diagnose bias-variance

- In high dimensions we cannot draw decision curves to inspect bias-variance.
- We calculate error values to infer the source of errors on the training set, as well as on the dev set.
- To determine bias, we need a base line, such as human level performance.

- Example:

Human level error	$\approx 0\%$			
Training error	0.5%	15%	1%	12%
Dev error	1%	16%	11%	20%
	low bias low variance	high bias	high variance	high bias high variance

Orthogonalization

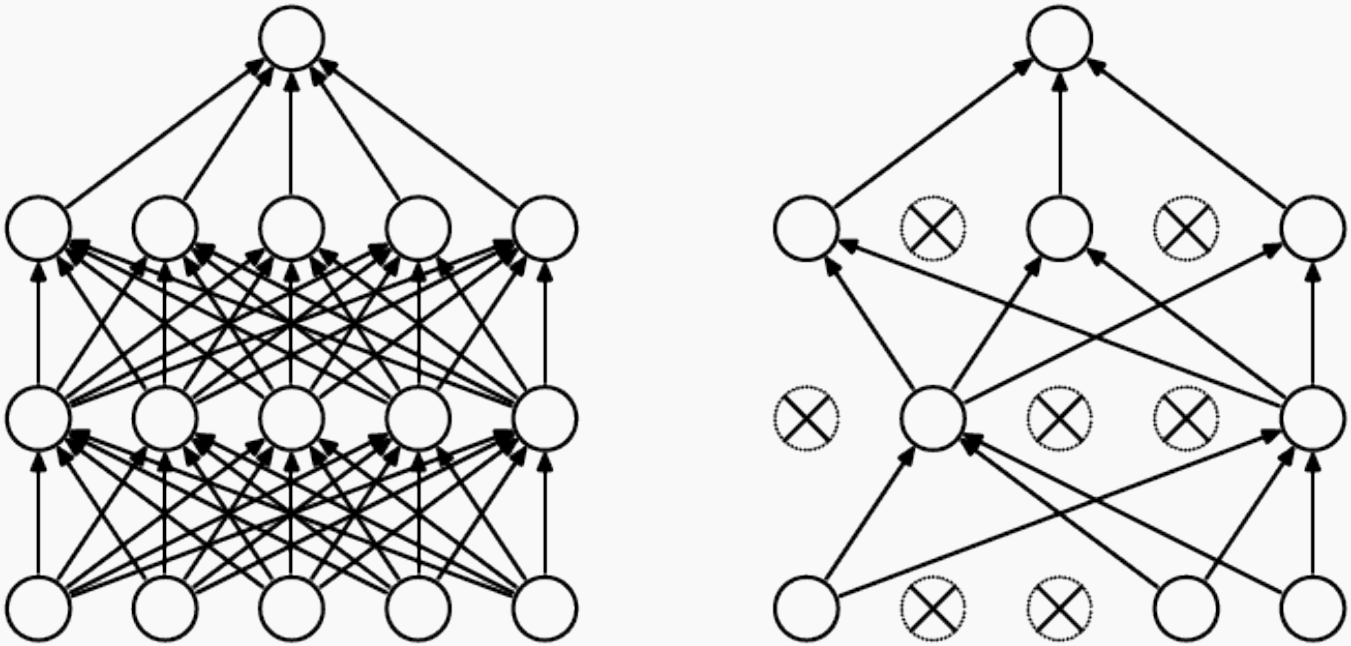


- Orthogonalization aims to decompose the process to adjust NN performance.
- It assumes the errors come from different sources and uses a systematic approach to minimize them.
- Early stopping is a popular regularization mechanism, but couples the bias and variance errors.

Dropout

Dropout

- Regularization technique for deep NN.
- The method is employed at training time, by eliminating the output of some units randomly (setting the output to zero).
- Can be used in combination with other regularization techniques (such as L2, batch normalization, etc.).



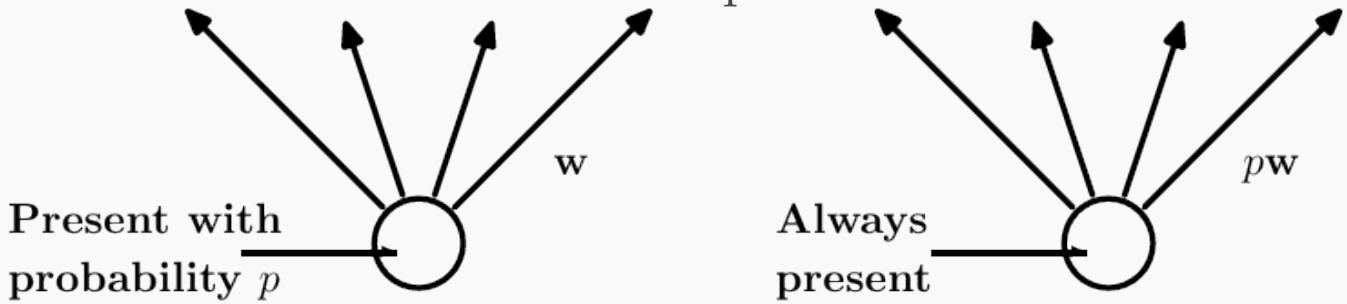
(a) Standard Neural Net

(b) After applying dropout.

Source: Hinton et. al

Motivation and direct implementation

- **Purpose**: prevent the co-adaptation of feature detectors for a set of neurons, and avoid overfitting.
  - It enforces the neurons to develop an individual role on their own given an overall population behavior.
  - Training weights are encouraged to be spread along the NN, because no neuron is permanent.
- **Interpretation**: training examples provide gradients from different, randomly sampled architectures.
- **Direct implementation:**
  - At training time: eliminate the output of some units randomly.
  - At test time: all units are present.



(a) At training time (b) At test time

Source: Hinton et. al

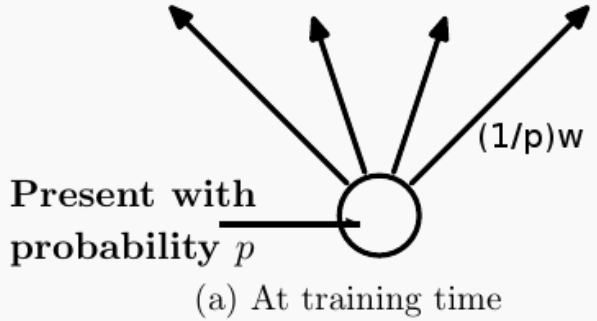
Inverted dropout

- Current implementations use ***inverted dropout***
  - Weighting is performed during training.
  - Does not require re-weighting at test time.
- In particular, for layer l,

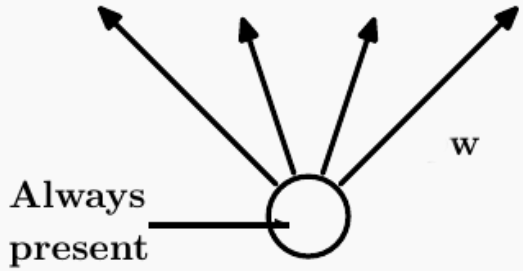
$$z^{[l]} = \frac{1}{p_l} W^{[l]} D^{[l]} a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g(z^{[l]}),$$

- Notation:

p_l : Dropout probability.
$D^{[l]}$: Dropout activations.
$a^{[l-1]}$: Output from previous layer.
$W^{[l]}$: Layer weights.
$b^{[l]}$: Offset weights.
$z^{[l]}$: Linear output.
$g(\cdot)$: Nonlinear activation function.



(a) At training time



(b) At test time

Understanding dropout

We aim to understand dropout as a regularization technique on simplified neural architectures such as:

- ***Linear networks.***
- ***Linear regression.***
- ***Logistic regression.***
- ***Deep networks.***

These results are are based on the following reference:

Pierre Baldi and Peter J Sadowski, "Understanding dropout," in *Advances in Neural Information Processing Systems*, 2013, pp. 2814–2822.

Dropout in linear networks

- **Linear network**: all activations units correspond to the identity function.
- For a single training example we get

$$z^{[l]} = W^{[l]} D^{[l]} z^{[l-1]}.$$

- The expectation over all possible network realizations:

$$\mathbb{E}\{z^{[l]}\} = p_l W^{[l]} z^{[l-1]},$$

- p_l corresponds to the probability of keeping a unit on layer l.
- The ensemble average replaces $W^{[l]}$ with $p_l W^{[l]} \rightarrow$ reduces its weight values.

Dynamics of a single linear unit

- Consider the error terms for the averaged ensemble network, and dropout (single example):

$$E^{\text{ens}} = (y^{(i)} - p_l W^{[l]} x^{(i)})^2$$
$$E^{\text{d}} = (y^{(i)} - W^{[l]} D^{[l]} x^{(i)})^2.$$

1. We minimize these cost functions with gradient descent.
2. Compute the gradients.
3. Take expectation over dropout realizations.
4. Obtain:

$$\mathbb{E}\{E^{\text{d}}\} = E^{\text{ens}} + \sum_{r=1}^{n_1} \frac{1}{2} \operatorname{var}(D^{[l]})(x_r^{(i)})^2 w_r^2$$

- Dropout corresponds to a regularized cost function of the ensemble network.

Dropout in linear regression

- Features $X \in \mathbb{R}^{m \times n}$; Observations $y \in \mathbb{R}^n$.
- Objective:

$$\min_{w \in \mathbb{R}^n} \quad \|y - Xw\|^2.$$

- With dropout: the input is transformed into $D \odot X$, where $D \sim \text{Bernoulli}(p)$.
- Operating over the previous expression we get

$$\min_{w} \quad \|y - pXw\|^2 + p(1-p)\|\Gamma w\|^2$$

  with $\Gamma = (\text{diag}(X^T X))^{-1/2}$.
- Change of variable $\tilde{w} = pw$

$$\min_{\tilde{w}} \quad \|y - X\tilde{w}\|^2 + \frac{1-p}{p}\|\Gamma \tilde{w}\|^2,$$

  with $\lambda = \frac{1-p}{p}$.

Dropout in logistic regression

- Single logistic unit with n inputs:

$$\sigma(z) = a^{[1]} = \frac{1}{1+e^{-z}} \text{ and } z = w^T x.$$

- The ***normalized weighted geometric mean*** over al possible network configurations corresponds to a feedforward pass of the averaged weights.

$$\text{NWGM} = \frac{G}{G+G'} = \frac{1}{1+e^{-\sum_j pw_j x_j}} = \sigma(pz).$$

- Definitions:
  - Total number of network configurations: $m = 2^n$.
  - $a_1^{[1]}, \ldots, a_m^{[1]}$ possible outcomes.
  - Weighted geometric mean: $G = \prod_i a_i^{P_i}$.
  - Weighted geometric mean of the complements $G' = \prod_i (1 - a_i^{[1]})^{P_i}$.

Dynamics of a single logistic unit

- The result from a single linear unit generalizes to a sigmoidal unit as well.
- The expected gradient of the dropout network:

$$\mathbb{E}\left\{\frac{\partial E^{\mathrm{d}}}{\partial w_i}\right\} \approx \frac{\partial E^{\mathrm{ens}}}{\partial w_i} + \lambda \sigma'(pz) x_i^2 \operatorname{var}(p) w_i.$$

- The expectation of the dropout gradient corresponds approximately to the gradient of the ensemble network plus a ridge regularization term.

Dropout in Deep Neural Networks

- Network of sigmoidal units.
- Output of unit i in layer h: $a_i^{[l]} = \sigma\Big(\sum_j W_{ij}^{[l]} a^{[l-1]}\Big)$
- ***Normalized weighted geometric mean***:

$$\mathrm{NWGM}(a_i^{[l]}) = \frac{\Pi_N (a_i^{[l]})^{P(N)}}{\Pi_N (1 - a_i^{[l]})^{P(N)} + \Pi_N (a_i^{[l]})^{P(N)}}$$

  where N ranges over all possible configuration networks.
- Averaging properties of dropout:

$$\mathbb{E}\{a_i^{[l]}\} = \sigma\Big(\mathbb{E}\Big\{\sum_j W_{ij}^{[l]} a_i^{[l-1]}\Big\}\Big)$$

- **Take-out message:** the expected dropout gradient corresponds to an approximated ensemble network, regularized by an adaptive weight decay with a propensity for self-consistent variance minimization.
- **Convergence** can be understood via analysis of stochastic gradient descent.

Batch normalization

Problems of deep networks

- Method of adaptive reparametrization, motivated by the difficulty of training very deep models.
- In gradient descent, parameters from all layers are updated at the same time.
  - composition of many functions can have unexpected results because all functions have been changed simultaneously.
  - learning rate becomes difficult to tune.
- Consider a linear network with a single neuron per layer and single input x.
- We update $w \leftarrow w - \epsilon g$, where $g = \nabla_w \hat{y}$:

$$\hat{y} \leftarrow (w^{[1]} - \epsilon g^{[1]})(w^{[2]} - \epsilon g^{[2]}) \dots (w^{[L]} - \epsilon g^{[L]})x.$$

- Previous update has many high order components, that can influence greatly the value of $\hat{y}$.

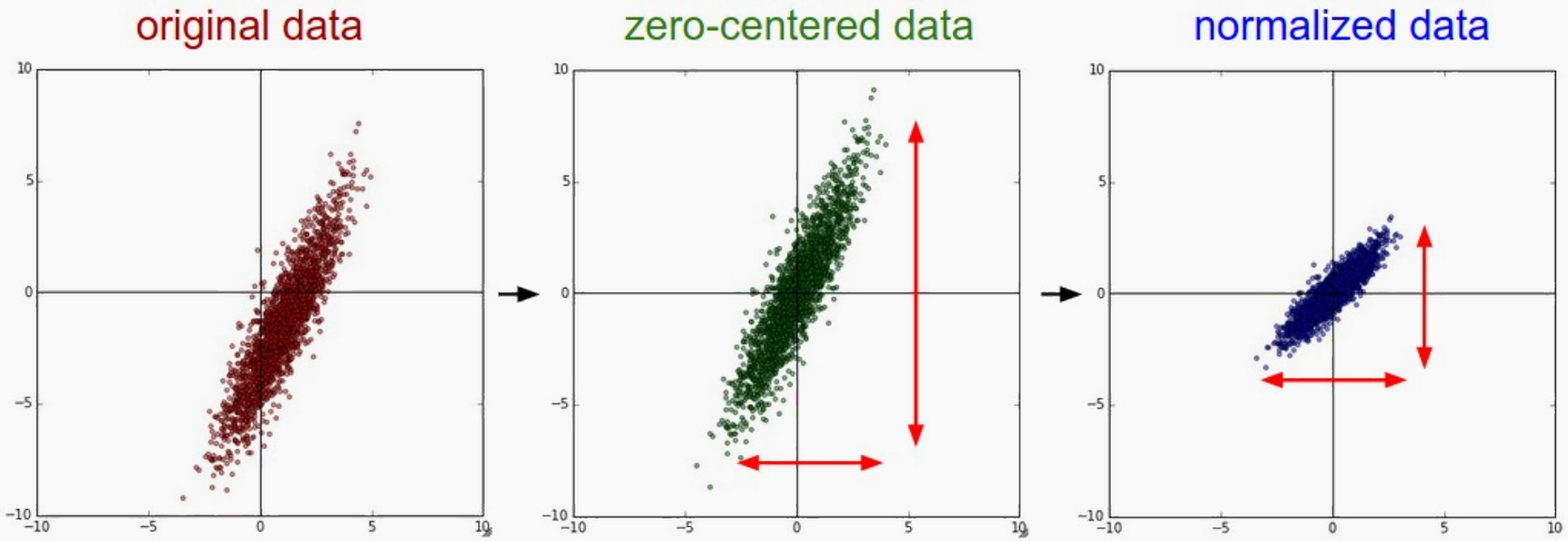
Input normalization

- The method is inspired by the normalization step normally applied to an input:

$$\widetilde{X}^{\{i\}} = \frac{X^{\{i\}} - \mu}{\sigma + \epsilon}$$

where $\epsilon = 10^{-8}$ is frequently used,

$$\mu = \frac{1}{m}\sum_r x^{\{i\}(r)}, \text{ and } \sigma^2 = \frac{1}{m}(x^{\{i\}(r)} - \mu)^2.$$



Batch normalization

- Batch normalization extends the concept to other hidden layers.

$$Z_{\text{norm}}^{\{i\}[l]} = \frac{Z^{\{i\}[l]} - \mu^{\{i\}[l]}}{\sigma^{\{i\}[l]} + \epsilon}$$

  where

$$\mu^{\{i\}[l]} = \frac{1}{m}\sum_r z^{\{i\}[l](r)}, \text{ and } (\sigma^{\{i\}[l]})^2 = \frac{1}{m}\sum_r (z^{\{i\}[l](r)} - \mu^{\{i\}[l]})^2.$$

- i refers to the mini-batch index; m to the number of elements.
  - the normalization depends on the minibatch.
- The outcome is rescaled with new parameters:

$$\widetilde{Z}^{\{i\}[l]} = \gamma^{\{i\}[l]} Z_{\text{norm}}^{\{i\}[l]} + \beta^{\{i\}[l]},$$

  where $\gamma^{\{i\}[l]}$ and $\beta^{\{i\}[l]}$ are incorporated in the learning process.

Batch normalization

- The scheme has the same expressive capabilities
  - setting $\beta^{\{i\}[l]} = \mu^{\{i\}[l]}$ and $\gamma^{\{i\}[l]} = \sigma^{\{i\}[l]}$.
- The weights from one layer do not affect the statistics (first and second order) of the next layer.
- The offsets $b^{[l]}$ become obsolete.
- **Testing:** a weighted average on all parameters:

$$\gamma_t = \alpha\gamma_t + (1-\alpha)\gamma^{\{i\}[l]}$$
$$\beta_t = \alpha\beta_t + (1-\alpha)\beta^{\{i\}[l]}$$
$$\mu_t = \alpha\mu_t + (1-\alpha)\mu^{\{i\}[l]}$$
$$\sigma_t = \alpha\sigma_t + (1-\alpha)\sigma^{\{i\}[l]}$$

Optimization algorithms

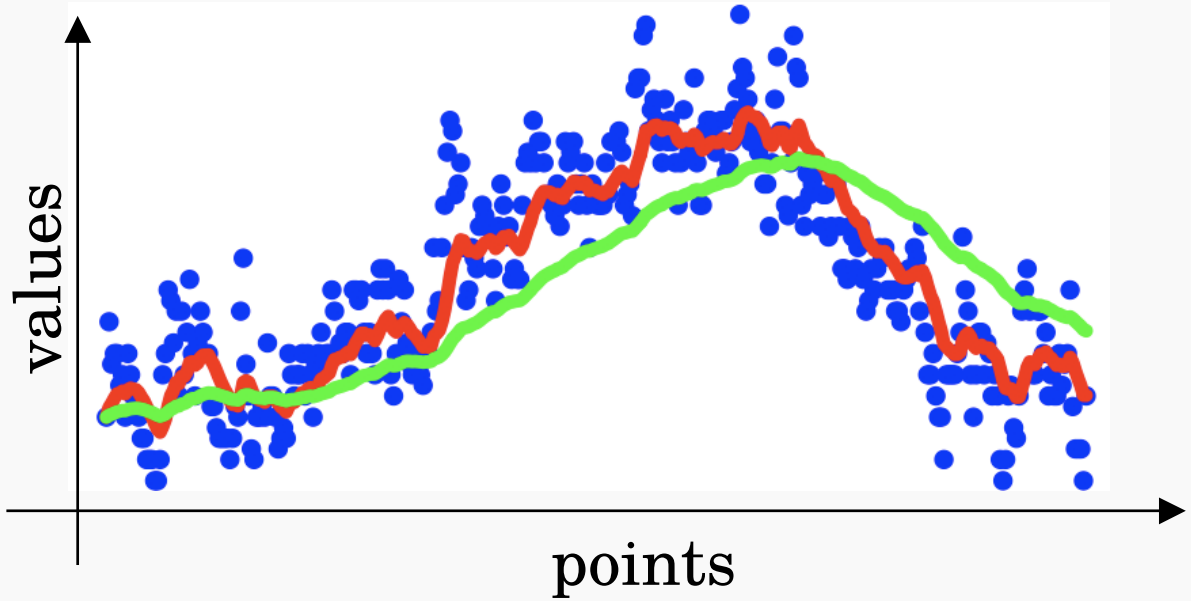
Exponentially weighted moving average

- Recursive equation to compute moving averages:

$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t,$$

  where θ_t is a data point, and v_t the moving average estimate.

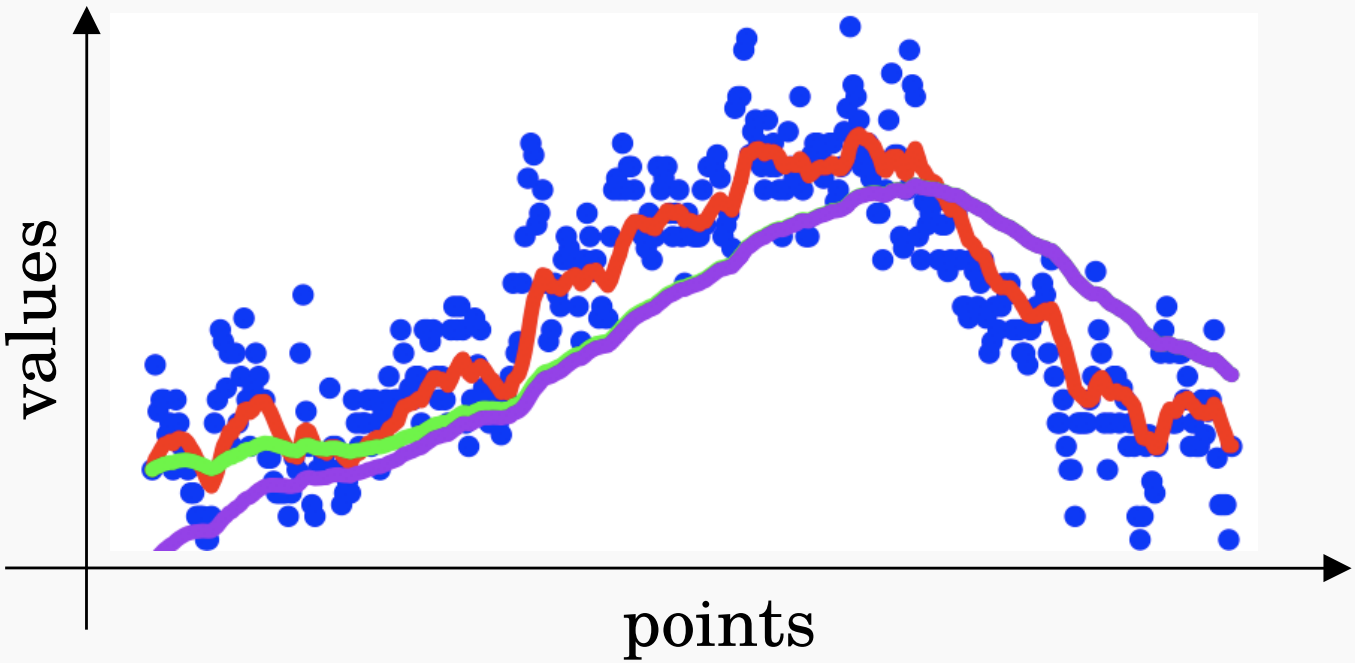
- β controls the amount of points to consider (variance):
  - **Rule of thumb:** $N = \frac{1+\beta}{1-\beta}$ amounts to 86% of influence.
  - $\beta = 0.9$ corresponds to 19 points.
  - $\beta = .98$ corresponds to 99 points (wide window).
  - $\beta = 0.5$ corresponds to 3 points (susceptible to outliers).



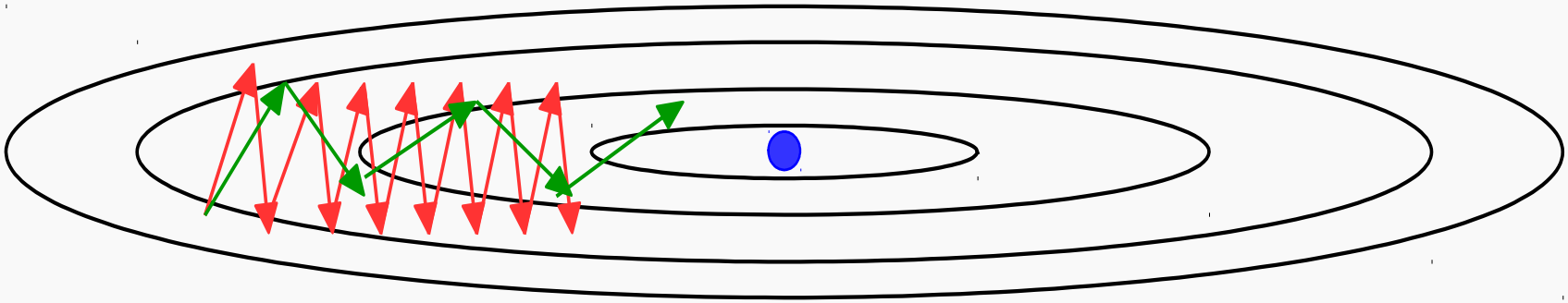
Bias correction

- The rule of thumb works for sufficiently large N.
- Otherwise, the first values are biased.
- We can correct the variance with:

$$v_t^{\text{corrected}} = \frac{v_t}{1 - \beta^t}.$$



Gradient descent



- The problem with gradient descent is that it will have high variance in some dimension if the problem is ill-conditioned.
- To improve the descent, we aim to estimate directions of high variance and reduce its influence.
- Descent with momentum, RMSprop or Adam, help reduce the variance and speed up convergence.

Gradient descent with momentum

- The algorithm:

  On iteration t:
  Compute dW, db on current mini-batch.
  $v_{dW} = \beta v_{dW} + (1 - \beta) dW$.
  $v_{db} = \beta v_{db} + (1 - \beta) db$.
  $W = W - \alpha v_{dW}$.
  $b = b - \alpha v_{dbW}$.

- Gradient with momentum performs an exponential moving average over the gradients.
- This will reduce the variance and give more stable descent directions.
- Bias correction is usually not applied.

RMSprop

- The algorithm:

  On iteration t:
  Compute dW, db on current mini-batch.
  $s_{dW} = \beta_2 s_{dW} + (1 - \beta_2) dW^2$.
  $s_{db} = \beta_2 s_{db} + (1 - \beta_2) db^2$.
  $W = W - \alpha \frac{dW}{\sqrt{s_{dW}} + \epsilon}$.
  $b = b - \alpha \frac{db}{\sqrt{s_{db}} + \epsilon}$.

- The algorithm performs an exponential moving average over the squared gradient components.
- $\epsilon = 10^{-8}$ controls numerical stability.
- High variance gradients will have larger values $\rightarrow$ the squared averages will be large $\rightarrow$ reduces the step size.
- Allows a higher learning rate $\rightarrow$ faster convergence.

Adaptive moment estimation (Adam)

- The algorithm:

  1: On iteration t for W update:

  2: Compute dW on current mini-batch.

  3: $v_{dW} = \beta_1 v_{dW} + (1 - \beta_1) dW$.

  4: $s_{dW} = \beta_2 s_{dW} + (1 - \beta_2) dW^2$.

  5: $v^{\text{corrected}} = \frac{v_{dW}}{1-\beta_1}$

  6: $s^{\text{corrected}} = \frac{s_{dW}}{1-\beta_2}$

  7: $W = W - \alpha \frac{v^{\text{corrected}}}{\sqrt{s_{dW}} + \epsilon}$.

- Same equations to update b.
- Adam estimates the moving average of gradients as well as its second momentum.
- It also incorporates bias correction.

Gradient checking

Gradient checking

- Useful technique to debug code of manual implementations of neural networks.
- Not intended for training of networks, but it can help to identify errors in a backpropagation implementation.
- Derivative of a function:

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon} \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}.$$

- The approximation error is in the order $O(\epsilon^2)$.
- In the multivariate case, the ϵ term affects a single component:

$$\frac{df(\theta)}{d\theta_r} \approx \frac{f(\theta_r^+) - f(\theta_r^-)}{2\epsilon}$$

where $\theta_r^+ = (\theta_1, \ldots, \theta_r + \epsilon, \ldots, \theta_n)$, $\theta_r^- = (\theta_1, \ldots, \theta_r - \epsilon, \ldots, \theta_n)$.

Algorithm for gradient checking

1: Reshape input vector in a column vector θ.
2: **for** each r component **do**
3: $\quad \theta_{\text{old}} \leftarrow \theta_r$
4: $\quad$ Calculate $f(\theta_r^+)$ and $f(\theta_r^-)$.
5: $\quad$ Compute approx. $\frac{df(\theta)}{d\theta_r}$.
6: $\quad$ Restore $\theta_r \leftarrow \theta_{\text{old}}$
7: **end for**
8: Verify relative error is below some threshold:

$$\xi = \frac{\|d\theta^{\text{approx}} - d\theta\|}{\|d\theta^{\text{approx}}\| + \|d\theta\|}$$