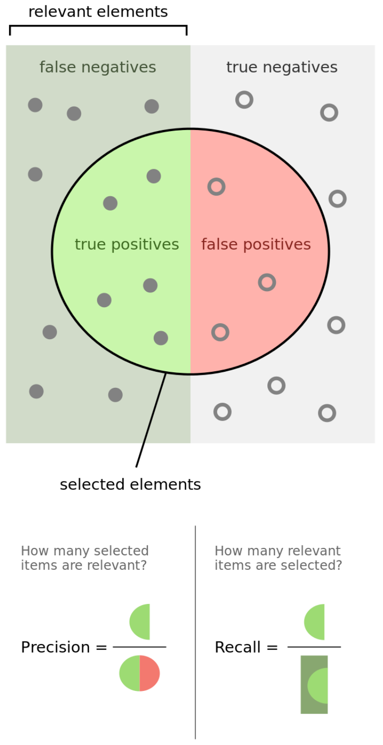
Formula

$$Ax = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}$$



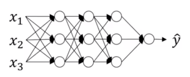
sigmoid: $\frac{1}{1+e^{-x}}$

sigmoid drv: $\sigma(1-\sigma)$

tanh drv: $1 - tanh^2(x)$

Batch Norm

Batch Normalization



$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$

$$z^{(i)}_{\text{norm}} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{\text{norm}} + \beta$$

- Mini-batch Batch Norm
- Usually applied before activation
- At test time: use running average of mean and var computed during train time

γ and β are learnable parameters at each layer.

(i) accelerates learning by reducing covariate shift, decoupling dependence of layers, and/or allowing for higher learning rates/ deeper networks, (ii) accelerates learning by normalizing contours of output dis- tribution to be more uniform across dimensions, (iii) Regularizes by using batch To receive full credit, these responses included three components: (i) Mini-batches might be small at test time. (ii) Smaller mini-batches mean the mini-batch statistics are more likely to differ from the mini-batch statistics used at training. (iii) Moving averages are better estimates.

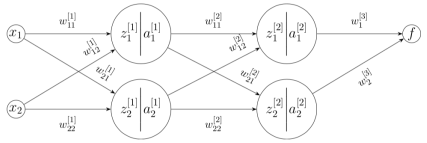
Optimizers

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

There are a few important differences between RMSProp with momentum and Adam: RMSProp with momentum generates its parameter updates using a momentum on the rescaled gradient, whereas Adam updates are directly estimated using a running average of first and second moment of the gradient. RMSProp also lacks a bias-correction term; this matters most in case of a value of 2 close to 1 (required in case of sparse gradients), since in that case not correcting the bias leads to very large stepsizes and often divergence,

Backprop



$$Z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad , \quad A^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \end{bmatrix}$$

$$Z^{[2]} = \begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} \quad , \quad A^{[2]} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[2]}) \\ \sigma(z_2^{[2]}) \end{bmatrix}$$

Solution:

(i) $\frac{\partial f(x)}{\partial z_1^{[2]}} = w_1^{[3]} \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} = w_1^{[3]}\sigma(z_1^{[2]})(1 - \sigma(z_1^{[2]})) = w_1^{[3]} a_1^{[2]}(1 - a_1^{[2]})$

$$f = \left[w_1^{[3]} \; w_2^{[3]}\right] A^{[2]}, \qquad A^{[2]} = \sigma(Z^{[2]})$$

(ii)

$$\frac{\partial f}{\partial Z^{[2]}} = \frac{\partial f}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} = \left[w_1^{[3]} \; w_2^{[3]}\right]^T \circ A^{[2]} \circ (1 - A^{[2]})$$

(iii) $\frac{\partial f(x)}{\partial Z^{[1]}} = \frac{\partial f(x)}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial Z^{[1]}} = \frac{\partial f(x)}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial Z^{[1]}} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} \end{bmatrix} \delta_2 \circ A^{[1]} \circ (1 - A^{[1]})$

(iv) $\delta_4 = \frac{\partial f(x)}{\partial w_{11}} = \frac{\partial f(x)}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial w_{11}} = \delta_3^T \begin{bmatrix} x_1 \\ 0 \end{bmatrix}$

GAN

Generative Adversarial Networks (GANs)

- GANs represent a game between two players:
  1. Generator G: producing samples hard to distinguish from real training samples
  2. Discriminator D: distinguishing G’s generated samples from real training samples
- GAN cost functions:
  1. Discriminator cost: $J^D = -\frac{1}{m_{\text{real}}}\sum_{i=1}^{m_{\text{real}}} \log\left(D(x^{(i)})\right) - \frac{1}{m_{\text{gen}}}\sum_{i=1}^{m_{\text{gen}}} \log\left(1 - D(G(z^{(i)}))\right)$
  2. Generator cost: a) Saturating cost $J^G = \frac{1}{m_{\text{gen}}}\sum_{i=1}^{m_{\text{gen}}} \log\left(1 - D(G(z^{(i)}))\right)$

     b) Non-saturating cost $J^G = -\frac{1}{m_{\text{gen}}}\sum_{i=1}^{m_{\text{gen}}} \log\left(D(G(z^{(i)}))\right)$

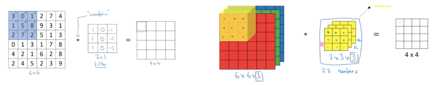
CNN

Convolutional Neural Networks (CNNs)

Convolution Layers:

Hyperparameters: Stride (s), Padding (p), Filter size (f)

- **2D (No depth)**
- **3D (e.g RGB channels)**



# learnable parameters = (f * f (+1 for bias)) * n_f

Output shape = (n-f+1) * (n-f+1) * n_f

# learnable parameters = (f * f * n_c (+1 for bias)) * n_f

Output shape = (n-f+1) * (n-f+1) * n_f

Convolutional Neural Networks (CNNs)

Padding and Strided Convolution Layers:

- **“Valid” convolutions ⇔ No padding:**

  Output shape = (n-f+1) * (n-f+1) * n_f

- **“Same” convolutions ⇔ Output shape should match Input shape**

  Output shape with padding = (n-f+1+2p) * (n-f+1+2p) * n_f

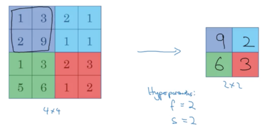
  p for “Same” convolutions = (f-1)/2

- **General formula for output shape:** $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$ **x n_f**

Convolutional Neural Networks (CNNs)

Pooling Layers:

Hyperparameters: Stride (s), Filter size (f)



- **General formula for output shape:** $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$ **x n_c**

Dropout

To do this, the authors suggest scaling the activation function by a factor of q during the test phase in order to use the expected output produced in the training phase as the single output required in the test phase. Thus:

Train phase: $O_i = X_i a(\sum_{k=1}^{d_i} w_k x_k + b)$

Test phase: $O_i = qa(\sum_{k=1}^{d_i} w_k x_k + b)$

A slightly different approach is to use **Inverted Dropout**. This approach consists in the scaling of the activations during the training phase, leaving the test phase untouched.

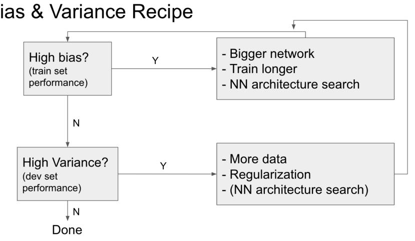
The scale factor is the inverse of the keep probability: $\frac{1}{1-p} = \frac{1}{q}$, thus:

Train phase: $O_i = \frac{1}{q} X_i a(\sum_{k=1}^{d_i} w_k x_k + b)$

Test phase: $O_i = a(\sum_{k=1}^{d_i} w_k x_k + b)$

Bias

Bias & Variance Recipe


High bias?
(train set performance)
Y
- Bigger network
- Train longer
- NN architecture search
N
High Variance?
(dev set performance)
Y
- More data
- Regularization
- (NN architecture search)
N
Done